# Estudio de las necesidades en las redes Ad Hoc y creación de un protocolo de enrutamiento

Sistemas Informáticos 2005-2006

**Ramón Cano Fernández**
**David Fernández Ortega**
**Mª Esther de la Osa Martínez**

Dirigido por:
Prof. Marta López Fernández
Dpto. Sistemas Informáticos y Programación
Grupo de Análisis, Seguridad y Sistemas (GASS)

**Los abajo firmantes: Ramón Cano Fernández, David Fernández Ortega, Mª Esther de la Osa Martínez, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y, mencionando expresamente a sus autores, tanto la presente memoria, como el código, la documentación, y/o el prototipo desarrollado.**

**Ramón Cano Fernández**
**David Fernández Ortega**
**Mª Esther de la Osa Martínez**

**Agradecimientos:**

La realización de este proyecto no hubiera sido posible sin el apoyo y dedicación de la directora de nuestro proyecto Marta López Fernández y del resto de integrantes del Grupo de Análisis, Seguridad y Sistemas (GASS) del Departamento de Sistemas Informáticos y Programación de la Universidad Complutense de Madrid, especialmente del director del Grupo, Javier García Villalba, y de Fabio Mesquita Buiati. Gracias por el seguimiento, asesoramiento y apoyo prestados a nuestro proyecto.

# INDICE

**Resumen del proyecto de sistemas informáticos 2005/2006**

# Estudio de protocolos para autoconfiguración y enrutamiento de redes ad-hoc y la implementación de un prototipo de enrutamiento

Nuestro trabajo en este proyecto se ha basado en un primer paso en estudiar las peculiaridades de las redes ad-hoc, sus características y su problemática, derivadas de ser un tipo de redes no estructuradas, en las que el movimiento de los nodos puede provocar alteraciones no deseadas en la red, también es necesario controlar la red para evitar duplicaciones de ips. Partimos la tesis de Fabio buiati la cual especificaba un protocolo de autoconfiguración para este tipo de redes, en un principio nuestro cometido era añadir seguridad, al protocolo de autoconfiguración y realizar el protocolo de sincronización para mantener la coherencia de la red, pero hicimos un estudio de este protocolo y descubrimos que antes de realizar estas tareas ,debíamos proporcionar una capa de red a los nodos, de tal manera que puedan comunicarse entre si.

Para poder establecer una red, necesitamos que se produzca enrutamiento entre los nodos de tal manera que cada nodo hace de gateway hacia los demás , así se podrán realizar comunicaciones entre todos los nodos. Realizamos un estudio de los distintos protocolos de enrutamiento aplicables a las redes ad-hoc, existen una gran cantidad de protocolos que se basan en centralizar el enrutamiento, estos protocolos no nos parecieron muy adecuados por que podrían tener problemas en redes muy inestables , por ello hemos elegido como protocolo de enrutamiento RIP V2 , y hemos implementado un prototipo basado en este protocolo de enrutamiento, el prototipo esta implementado en c en sistema operativo linux por coherencia con lo ya implementado. Hemos necesitado utilizar las librerías libpcap, para realizar el proceso de escucha de paquetes, libnet para construir los paquetes y posix threads ya que es fundamental que se ejecuten simultáneamente los procesos de escucha de paquetes, de gestión de las tablas y envío de las tablas actualizadas a los nodos vecinos. Nuestra aplicación tiene 3 partes fundamentales, el modulo de gestión de la información que es el encargado de gestionar las estructuras de datos que formaran las tablas de enrutamiento, así como el encargado de procesar la información de los paquetes recibidos de los nodos vecinos y enviar las tablas actualizadas a los vecinos, el modulo de transmisión de información que es el encargado de recibir los paquetes involucrados en el protocolo y escribirlos en un archivo para su posterior proceso, así como de enviar los paquetes para la actualización de las tablas, entre estos dos módulos se encuentra el modulo de codificación/descodificación que hará de interfaz entre los dos módulos principales.

**Summary of the sistemas informáticos project 2005/2006**

# Autoconfiguration protocols study for ad-hoc networks and a implementation of a prototype of routing protocol

Our work in this project has been based, in the study of the peculiarities of the ad-hoc networks, its specifications and problems, which are caused because this kind of networks aren't structured networks, in this networks the movement of the nodes could cause a bad performance of the network, to avoid duplicates ips we need a control protocol. We start with the Fabio buiati Tesis autoconfiguration protocol, first we study this protocol, our work was add security and synchronization to the fabio's protocol, after our protocol study we decided that was more important create a network layer between nodes, in order to obtain this objective we have to create a routing module, by this way with the roting module, all network nodes can communicate among them.

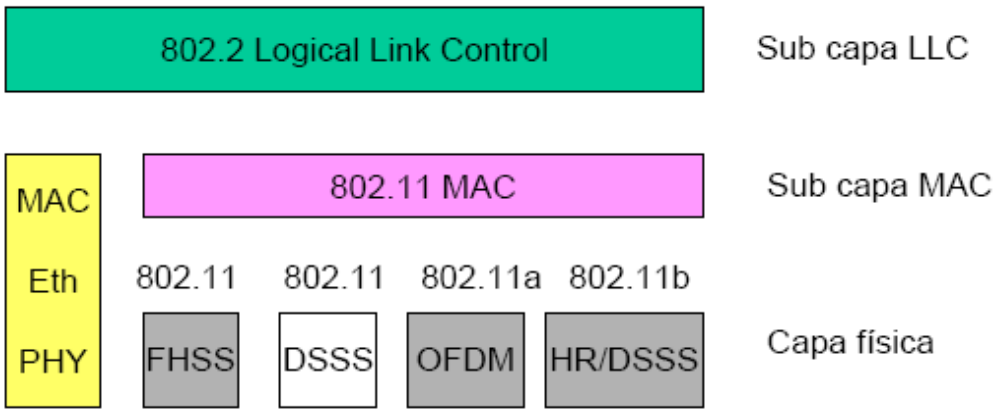
We need to make routing betwen nodes, in this way all node have the gateway role, there are a lot of ad-hoc networks routing protocols, but there are many protocols which centralize the routing in a few nodes, we think that these centralized protocols are not adapted for ad-hoc networks because the ad-hoc networks aren't stables, by this we decided to implement a routing protocol based on RIP V2, we've used c language and Linux to implement our prototype, We've needed c libraries to build our prototype these libraries are, lipcap to receive our protocol packets, libnet to build and send our protocol packets,and posix threads to make concurrent the receive and the process of the packets, our aplication has three modules, the management information module, the transmisión information module, and the codification/decodification module which conect the main modules.

# 1 Introducción

Las redes inalámbricas se han extendido rápidamente gracias a su característica principal, que es la movilidad; la posibilidad de no tener la restricción a un puesto fijo permite flexibilidad a la hora de crear la infraestructura de la red, es decir, tenemos una estructura cambiante y flexible, a diferencia de la estructuras fijas de la redes cableadas. Estas redes se suelen insertar dentro de redes cableadas para tener una estructura híbrida.

Una vez que hemos mostrado la gran ventaja de las redes inalámbricas vamos a pasar a desgranar su estructura y dar una descripción técnica de su arquitectura. Para este tipo de redes la arquitectura se basa en el estándar 802.11 del IEEE, este estándar define una capa LLC (Logical Link Control) que logra estandarizar la comunicación con capas superiores, ya que en las capas inferiores (la capa física) existe un número importante de interfaces y protocolos y necesitamos un método para poder aislar unas capas de otras. Dentro de la primera capa, la capa física, existe una especificación para cada tipo de medio, y al tener las ondas de radio como medio para la comunicación la capa física se separa en dos subcapas:

- La capa PLCP: capa para el realizar el procedimiento de convergencia, es decir, realiza el proceso de convergencia de los frames MAC sobre el medio.
- La capa PMD: capa dependiente del medio y se encarga de transmitir el frame.




A nivel físico tenemos que la comunicación se realiza dentro de tenemos los siguientes alternativas:

- 802.11a: Banda de 5 Ghz. Velocidad hasta 54 Mbps.
- 802.11b: Banda de 2,4 Ghz. Velocidad hasta 11 Mbps.
- 802.11g: Banda de 2,4 Ghz. Velocidad hasta 54 Mbps.

La transmisión de las ondas de radiofrecuencia se realiza a una frecuenta dentro de la banda ISM (Industrial, Scientific and Medical), en el siguiente gráfico podemos ver el rango de frecuencias de esta banda:

| Banda | Rango de Frec. |
|---|---|
| UHF ISM | 902 – 928 MHz |
| Banda S | 2 – 4 GHz |
| Banda S ISM | 2.4 – 2.5 GHz |
| Banda C | 4 – 8 GHz |
| Banda C ISM | 5.725 – 5.875 GHz |

Debido a las frecuencias en las que trabajamos tenemos el riesgo de interferencias debido a múltiples factores, en particular a las ondas de microondas que funcionan en frecuencias de 2,4 GHz.

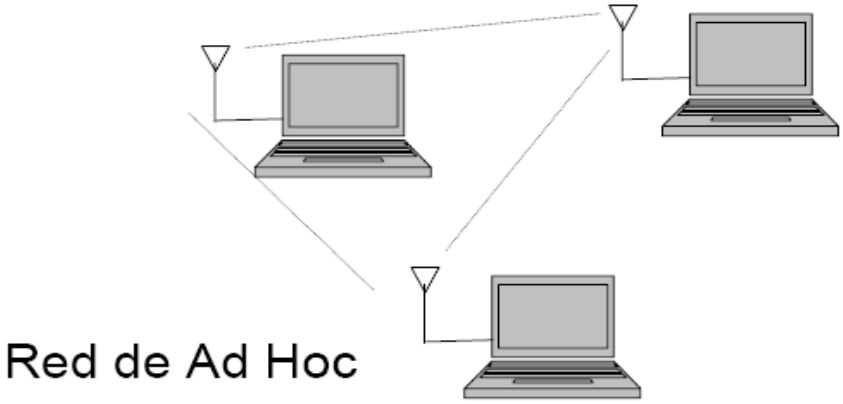
Una vez que tenemos la arquitectura del estándar 802.11 vamos a enumerar los componentes que puede poseer una red inalámbrica, estos son:

- Estaciones móviles: son los clientes iniciales y finales de la comunicación. Son los hosts del sistema, su principal diferencia con los hosts en redes cableadas es la capacacidad de movilidad que tienen.
- Access Point: componente que aparece en las redes de infraestructura; sirve para poder comunicar la red inalámbrica con una red cableada.
- Red de distribución: estructura que conecta los distintos hosts; aparece dentro de las redes de infraestructura.

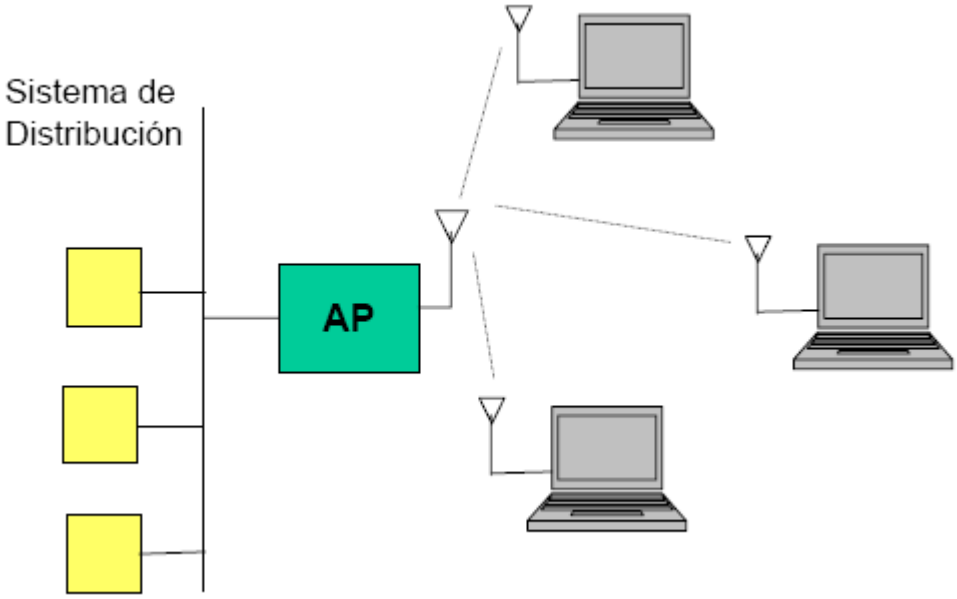
Otro concepto importante dentro de las redes inalámbricas es el bloque constructivo de la red, que es llamado conjunto básico de servicios (BSS, Basic Service Set); este conjunto básico de servicios se compone de las estaciones que se encuentran en disposición de realizar una comunicación; está comunicación se realiza dentro de un área básica de servicios, que es el área en la cual la comunicación es posible, una carácteristica de esta área es que sus límite son difusos.

Este conjunto básico de servicios puede ser de dos tipos:

- Redes ad hoc o independientes: redes donde la comunicación la realizan los propios hosts, estos son capaces de enrutar los paquetes. El componente único y principal de estas redes son los hosts móviles, no existen ni puntos de acceso ni una red de distribución.




- Redes de infraestructura: redes donde existe una comunicación a una red cableada; los componentes de este tipo de redes son los hosts, la red de distribución y los access point.




Las redes ad hoc o independientes las comentaremos más detalladamente en el próximo punto, pero aquí esbozaremos el tipo de redes de infraestructura.

**Redes de infraestructura**

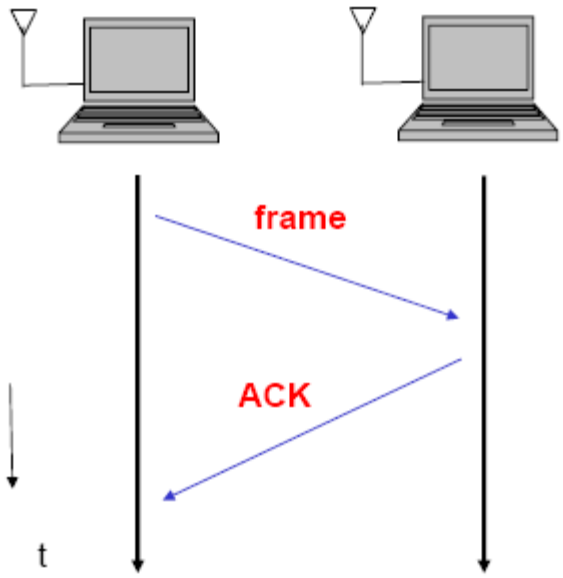
Este tipo de redes utilizan un punto de acceso para la comunicación entre dos hosts, los paquetes son enviados al punto de acceso y este los reenvia al nodo destino. Debido a las restricciones existentes con los BSS para cubrir áreas extensas, el protocolo 802.11 extiende el área de servicios gracias al conjunto extendido de servicios (ESS) y dentro de este nuevo conjunto de servicios las estaciones se comunican entre sí; dentro de los nuevos servicios

tenemos que los puntos de acceso pueden funcionar como bridges para las comunicación con otras redes.

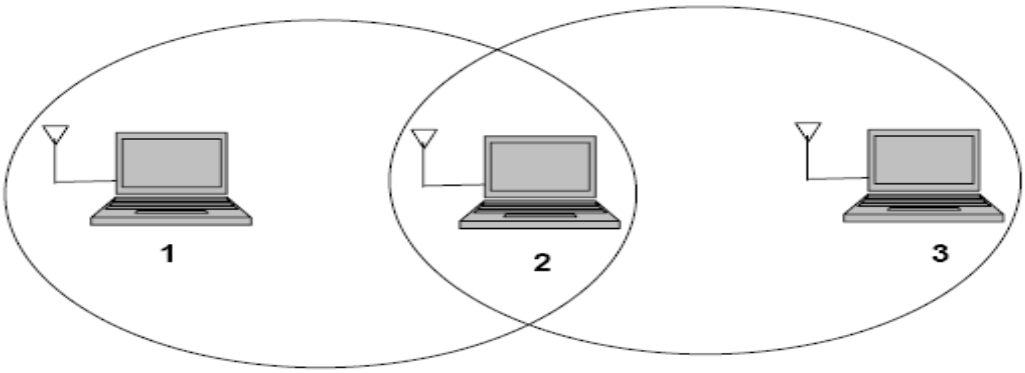
Un componente importante de este conjunto extendido de servicios es el sistema de distribución, que se compone del punto de acceso y de la red ethernet, este sistema se encarga de actualizar la ubicación física de las estaciones y despachar correctamente los marcos.

Dentro de este tipos de redes, una capa importante para poder obtener una red con óptimos niveles de rendimiento, es la subcapa MAC, que se pude optimizar su rendimiento gracias a la posibilidad de parametrización que tienen. Esta capa utiliza CSMA, para lograr el acceso al medio de transmisión, pero a diferencia que Ethernet las colisiones no se detectan sino que se evitan, el protocolo de acceso al medio es distribuido.

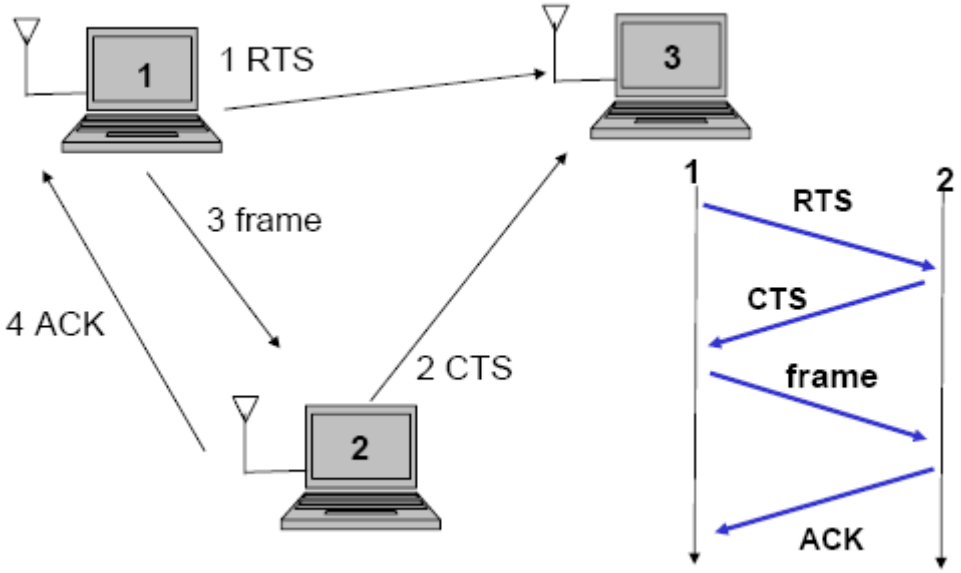
Para la transimisión de paquetes se utiliza una banda de frecuencia sin licencia llamada ISM; esta transmisión tiene múltiples fuentes de interferencia y por ello se realiza una comunicación donde cada frame tiene su ACK.



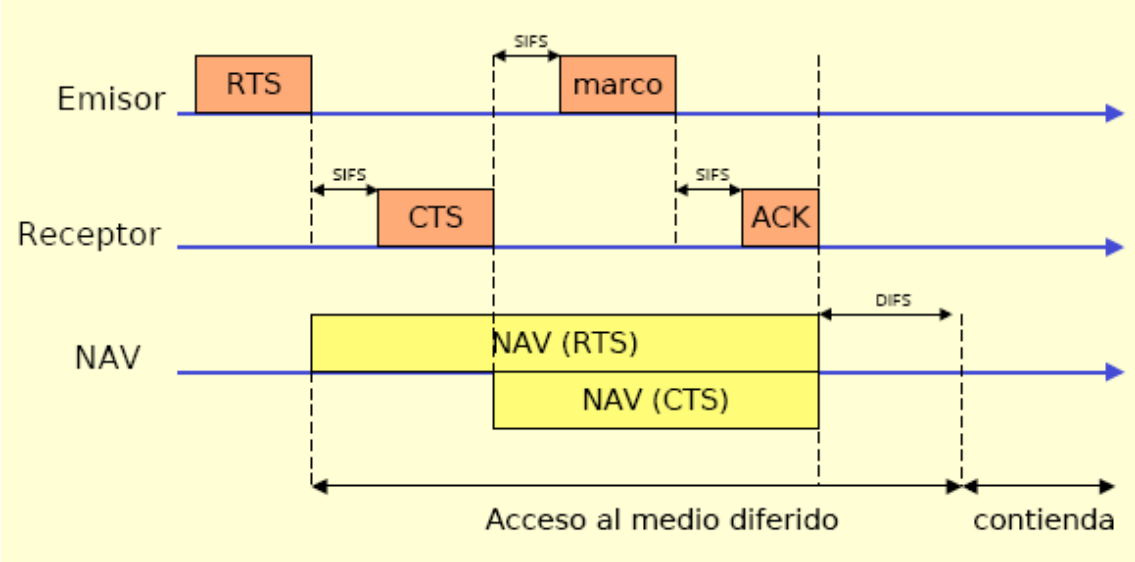

Un problema existente dentro de la comunicación entre hosts es la existencia de nodos ocultos, por tanto puede haber colisiones debidas al envió de frames por parte de estos dos nodos debido a que el medio utilizado para la transmisión es half-duplex.




Para solucionar este problema tenemos el procedimiento RTS/CTS; este procedimiento consiste en el envió por parte de la estación transmisora un RTS, donde dentro de este frame tenemos un campo del tiempo total de transmisión (NAV). La estación receptora contesta con CTS, y envia el tiempo que es necesario silenciar el área para evitar colisiones, así evitamos colisiones entre nodos que son ocultos entre si.

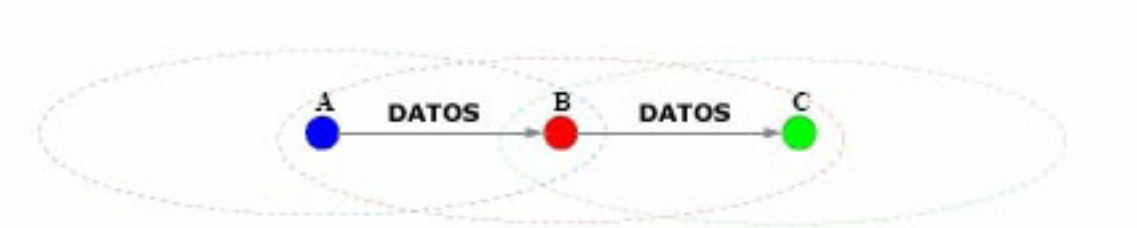



Los modos de acceso al medio se basa en dos formas bien diferenciadas; por un lado tenemos una función de coordinación distribuida (DCF), donde la base es el CSMA/CA, que comprueba si el enlace de radio esta siendo usado antes de emitir, y, en el caso de que este siendo usado, utiliza un tiempo de espera aleatorio tras cada marco. Por otro lado tenemos la función punto de coordinación (CPF), donde la inclusión de un punto de acceso asegura que el medio se utiliza sin contienda. Además podemos utilizar un vector de reserva de la red (NAV), que consiste en insertar un campo dentro del marco 802.11 para reservar durante el tiempo el medio de transmisión, este campo indica el tiempo en el que medio permanecerá ocupado, y de este modo se consiguen las operaciones atómicas (Send/ACK, RTS/CTS).


Emisor
RTS
SIFS
marco
Receptor
SIFS
CTS
SIFS
ACK
NAV
DIFS
NAV (RTS)
NAV (CTS)
Acceso al medio diferido
contienda

# 2 Redes Ad Hoc / Manet

Son redes inalámbricas sin infraestructura, sin puntos de acceso, donde cada nodo actúa simultáneamente como cliente o servidor de la red. El hecho de usar transmisión inalámbrica influye en el comportamiento de la red. Las comunicaciones tienen un rango de transmisión limitado, en el que el receptor es capaz de recibir e interpretar correctamente la señal que envía el emisor, si el receptor se encontrara fuera del rango de alcance del emisor, no podrá interpretar correctamente los paquetes del envío. Por eso en las redes MANET todos los nodos colaboran para enviarse la información enrutando los paquetes de datos salto a salto, son lo que se denominan redes inalámbricas multi-salto.




Estas redes pueden considerarse como redes aisladas, o bien, como extensiones de redes fijas a las que están conectadas. Este último caso se conoce como redes ad hoc híbridas y necesitan de pasarelas entre ambas redes.

## 2.1 Características

La naturaleza dinámica de este tipo de redes, y el uso de un medio de transmisión inalámbrico, obligan a la consideración especial de sus características a la hora de decidir su implantación.

**Topología dinámica**

Todos los nodos que forman parte de la red son elementos móviles, lo que provoca que la topología de la red esté en constante cambio, y que los enlaces entre los nodos se creen y se destruyan dinámicamente.

**Ancho de banda limitado**

El ancho de banda disponible en una interfaz de red inalámbrica es inferior a la de una red cableada, además se ve infrautilizada debido a interferencia de señales y atenuación propias del medio.

A pesar de esto, el usuario, pretende continuar usando el mismo tipo de aplicaciones, o bien, aplicaciones con un coste similar.

**Consumo de energía**

Con frecuencia los nodos de la red estarán siendo alimentados por baterías, este hace necesario que el ahorro de energía sea algo importante a tener en cuenta.

**Seguridad**

Los problemas de trabajar con medio compartido, al que cualquiera puede tener acceso, acentúan los riesgos de seguridad de la red.

- Deben protegerse la confidencialidad de los datos, para que no sean utilizados por terceros no autorizados.
- Debe evitarse la autenticidad de los nodos pertenecientes a la red, puesto que un nodo podría simular ser un nodo confiable de la red.
- Además los ataques de Denegación de Servicio (DoS) son más sencillos de llevar a cabo en un entorno no controlado.

De estas características, deducimos que el diseño de protocolos para el enrutamiento de información en redes Ad Hoc, deben reaccionar ante los cambios topológicos de los nodos, mediante la creación de nuevas rutas. Además, tendrán en cuenta que el excesivo envío de mensajes de control, disminuye el ancho de banda disponible para el tráfico útil de información, y serán protocolos intensivos en procesamiento para evitar el agotamiento de las fuentes de alimentación de los nodos.

Además, deberán de implementar los mecanismos de seguridad pertinentes para evitar o contrarrestar los problemas de seguridad comentados previamente.

## 2.2 Arquitectura del Nodo


### 2.2.1 *Nivel de Enlace*

Como las redes inalámbricas utilizan un medio compartido que transporta los datos que se emiten, es necesario que los nodos no interfieran en las comunicaciones que puedan estar manteniendo los demás, por eso el control de acceso al medio (MAC) es uno de los aspectos más importantes. Por lo general se usan dos esquemas de coordinación:

- Centralizado, en el que existe un árbitro que va asignando el turno de palabra a cada emisor, de manera que ningún nodo puede transmitir fuera de su turno.
- Por contienda, en el que no existe un control centralizado. En este esquema cualquier nodo puede transmitir cuando considere

oportuno, sin embargo, es susceptible a colisiones, y cada nodo debe ser capaz de recuperase.

En el tipo de redes que estudiamos, se hace totalmente inviable un esquema de coordinación centralizado.

La capa MAC del 802.11 utiliza un acceso por contienda llamado CSMA/CA (Carrier Sense Multiple Access/Collision Avoidande) en el que un nodo no emite nada si detecta que canal está siendo usado para otra transmisión por otro nodo. Además antes de transmitir se encargará de generar mensajes Request To Send / Clear To Send, para que los nodos cercanos detecten que el canal va a ocuparse.

A pesar de ser el protocolo más ampliamente usado en el desarrollo de redes MANET, su uso presenta una serie de inconvenientes. El principal inconveniente es que ante una red con carga elevada se vuelve ineficiente, debido a que el rango de interferencia de cada nodo es elevado y una comunicación puede dañar otra que se esté llevando a cabo en otra parte de la red. Una de las ideas, adecuada al desarrollo de redes Ad Hoc, es minimizar la potencia de la señal, con la que se transmite, de manera que esta afecte al menor número de nodos posibles, o bien, utilizar antenas direccionales en la dirección del nodo destino.

### 2.2.2 *Nivel de Red*

La capa de red es la encargada de formar la MANET y enviar los paquetes de datos a sus destinos correctos. El motivo de hacerlo así, no es otro que hacer que la red Ad Hoc sea independiente de las tecnologías de acceso al medio utilizadas.

Los protocolos ya existentes para trabajar a este nivel, pueden sufrir algunas modificaciones para trabajar con los nodos Ad Hoc, de esta manera los nodos pueden comunicarse, unos con otros, de una manera transparente para las aplicaciones.

En una red de este tipo cada nodo debe comportarse como un router, manteniendo individualmente las rutas a otros nodos. Esto implica que cuando crezca el número de nodos en la red, crecerán el número de rutas que ha de mantener cada nodo, y por consiguiente, las tablas con la información de enrutado hacia otros nodos.

La escalabilidad dependiendo del número de nodo y la movilidad de estos, se convierten en los grandes problemas del mantenimiento de este tipo de redes, produciendo un aumento de la carga para el mantenimiento de las rutas, consumiendo el ya de por sí escaso ancho de banda.

Esto hace que los protocolos de enrutado usados para redes fijas no sean adecuados para MANET´s.

### 2.2.3 *Nivel de Transporte*

El protocolo de transporte UDP ofrece un servicio no confiable y no orientado a conexión de entrega de paquetes. Los mensajes de datos del nivel de aplicación son encapsulados en paquetes UDP y enviados confiando que alcanzarán correctamente su destino, aunque no se garantiza nada. Debido a su simpleza su aplicación en el tipo de redes que estudiamos es directa y no presenta inconvenientes.

### 2.2.4 *Nivel de Aplicación*

Para un uso comercial, se hace imprescindible que los protocolos usados en este nivel sean los mismos empleados para el mismo propósito en redes fijas; aunque para uso militar o de emergencias es posible que se ideen otros protocolos que atiendan más a las características de una red MANET.

# 3 Protocolo de Autoconfiguración

En este punto vamos a tratar todas las funcionalidades del protocolo existente para el cual se ha implementado el enrutamiento de la red. Todas las consideraciones que se han tenido en cuenta a la hora de hacer el desarrollo, atienden al funcionamiento específico que se comenta a continuación.

## 3.1 Objetivo

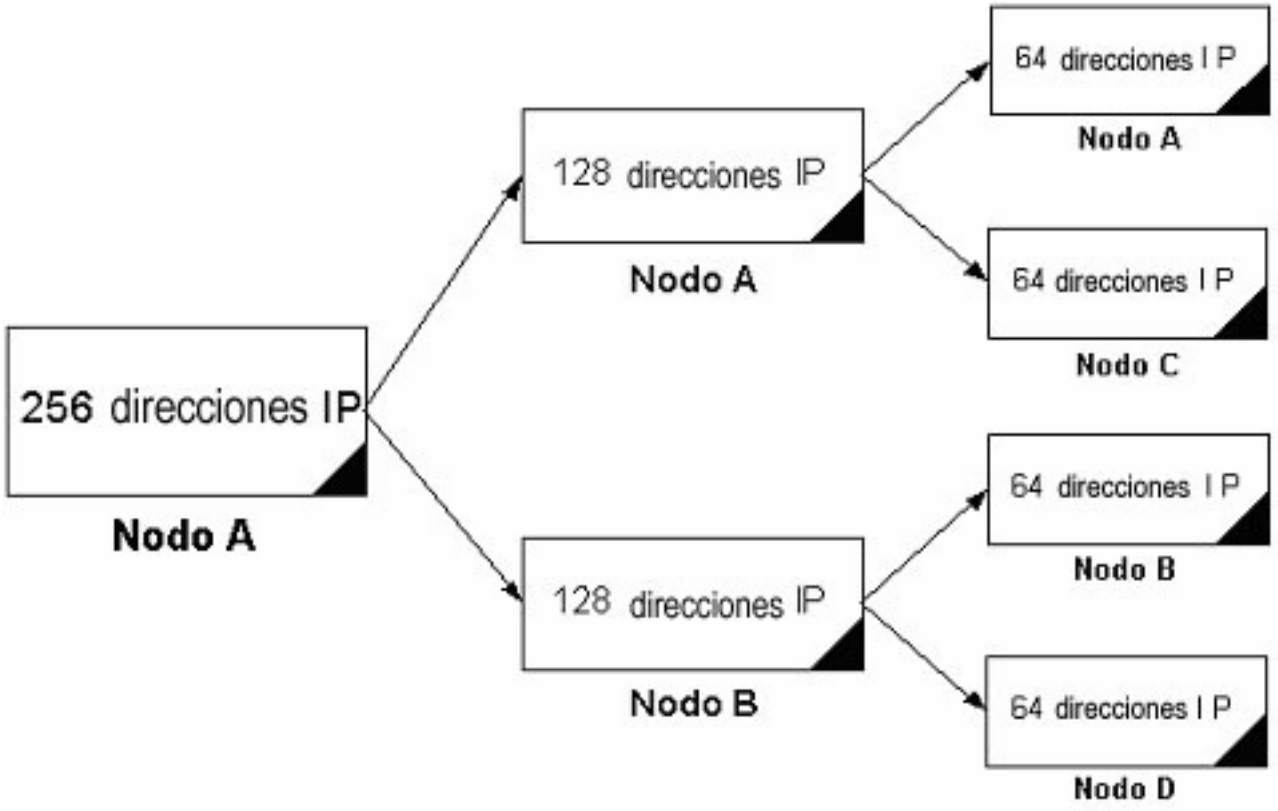
El objetivo principal del protocolo es asociar de forma segura y confiable una dirección IP a un nodo que desea unirse a la red ad hoc. Fueron analizadas las necesidades básicas de seguridad y definidas las modificaciones en el protocolo para hacerlo seguro. Como modificación principal, el uso de la autenticación para cada mensaje intercambiado entre los nodos utilizando el mecanismo de autoridades certificadoras distribuidas, esto garantiza que ningún nodo intruso puede crear mensajes o modificar los ya existentes con la intención de degradar el rendimiento del protocolo de autoconfiguración o hacer el servicio no disponible.

## 3.2 Funcionamiento

Inicialmente, llamaremos *nodo cliente* a aquel que desea unirse a la red y obtener una dirección IP, y *nodo servidor* al que atenderá a la solicitud.

Cada nodo válido y confiable perteneciente a la red posee una dirección IP identificando su interfaz y un conjunto de direcciones libres que llamaremos de free_ip_block, de entre las cuales seleccionara una dirección IP para servir a los nodos clientes que desean asociarse a la red. Dentro de una misma red ad hoc, los conjuntos de direcciones IP libres (free_ip_block) de los nodos deben ser disjuntos, garantizando así que dos o más nodos servidores no suministren la misma dirección IP para nodos clientes. Además de eso, cada partición lógica, o sea, cada red ad hoc posee un identificador único llamado Partition_id-PID. Así, todos los nodos que poseen la misma PID son parte de la misma red ad hoc, este identificador facilita la detección de unión y separación de redes ad hoc.

El protocolo DCDP (Dynamic Configuration Distribution Protocol) es un protocolo para la distribución de configuraciones de red como: dirección IP, máscara de red y gateway patrón basado en el modelo llamada “buddy system”, que utiliza el mecanismo de división binaria para suministrar conjuntos disjuntos de direcciones IP a los nodos de la red. La siguiente figura muestra el proceso de división binaria para el suministro de dirección IP a nuevos nodos:




Funcionamiento del modelo de división binaria.

Inicialmente, el nodo A posee todo el rango de direcciones IP (256 direcciones IP), teniendo en cuenta que una de esas direcciones es la que está asociada a la interfaz de red del nodo (free_ip_block con 255 direcciones IP). Aplicando los métodos de división binaria, el nodo A responde a una solicitud de configuración del nodo B y divide su conjunto de direcciones IP libres en dos mitades, suministrando la mitad para el nodo B y quedándose con la otra mitad. Después de esta división, el nodo A y el nodo B se quedan cada una con 128 direcciones, siendo una dirección IP para su interfaz de red y 127 para servir a otras peticiones. A partir del mismo proceso, el nodo A divide su conjunto de direcciones IP libres en dos mitades nuevamente suministrando la mitad para el nodo C, quedándose con la otra mitad. El mismo procedimiento ocurre con el nodo B que para atender una petición suministra la mitad de sus direcciones IP para el nodo D y se queda con la otra mitad. Este mismo proceso es usado para atender todas las nuevas peticiones hasta que todas las direcciones IP que pertenezcan a la free_ip_block de algún nodo sean agotadas.

Aceptando este modelo de distribución de direcciones, se presentan a continuación todos los posibles casos de uso posibles, debido a la topología dinámica de las redes ad hoc.

### 3.2.1 *Inicialización de la red ad hoc*

De acuerdo con el modelo de seguridad que el protocolo mantiene, la inicialización de la red ad hoc está íntimamente ligada al modelo de confianza colaborativa “K-out-of-N”, pueto ques es necesario que existan por lo menos K

nodos previamente configurados en la red para que nuevas peticiones puedan ser atendidas y autenticadas de forma segura. Por lo tanto, en la fase de inicialización de la red, el principal objetivo es el de inicializar los primeros K nodos que posteriormente atenderán las peticiones de nuevos nodos.

Para inicializar estos primeros K nodos, cada uno con su reparto de la clave secreta, y utilizando un nodo líder para poseer el certificado total asignado como clave SK para que pueda suministrar los repartos de la clave secreta SK. Vale resaltar, que una red ad hoc puede tener decenas o hasta centenas de nodos y, poseer apenas un nodo para hacer el suministro de las claves parciales no es una tarea fácil. Además de eso, mantener apenas un nodo para la realización de este servicio, así como para el almacenamiento de la clave secreta SK, se puede tonar un punto de fallo en la red por permitir que nodos adversarios unan sus esfuerzos y obtener la clave secreta a través de ataques de fuerza bruta.

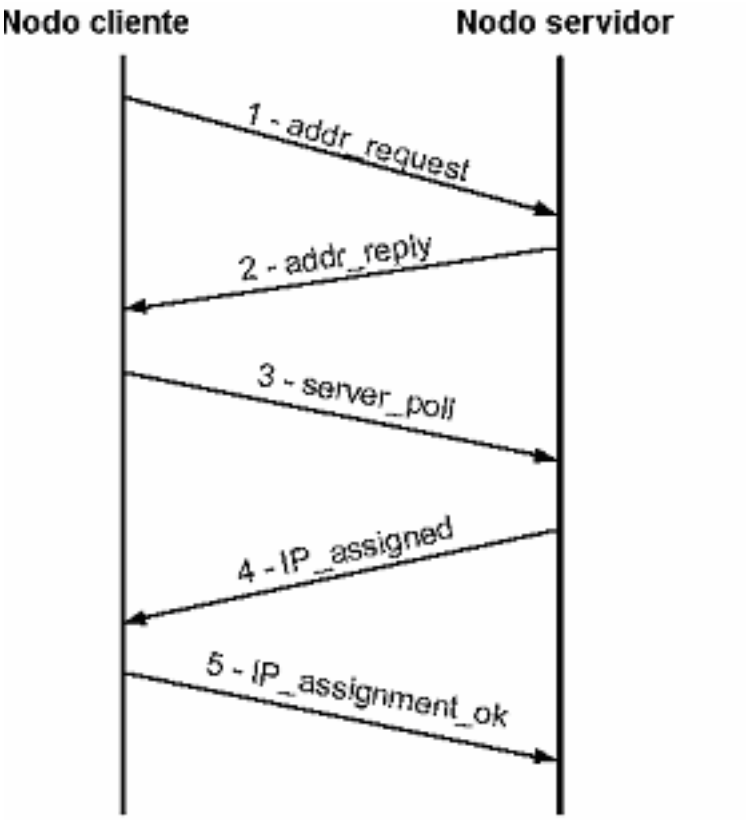
Por lo tanto, en el proceso de inicialización, el nodo líder inicia los primeros K nodos e inmediatamente después, este mismo nodo deja la red para evitar que algún nodo enemigo consiga obtener la clave secreta, garantizando así una inicialización segura y colaborativa a través del proceso de difusión.

A partir de este momento, el nodo posee un certificado confiable por todos los otros nodos pertenecientes a la red. Entretanto, el nodo aún no posee una dirección IP, lo que sólo el posible después de este proceso de obtención del certificado confiable. Así, sigue en la sección siguiente el proceso de asociación de una dirección IP a un nuevo nodo que desea unirse a la red.

### 3.2.2 *Solicitud de una dirección IP por parte un nuevo nodo*

Antes de que a un nuevo nodo se le asigne una dirección IP dentro de la red es necesario que haya obtenido el certificado válido siguiendo el modelo de confianza colaborativa.

Suponiendo que para la asociación de una dirección IP a un nodo, este posea un certificado válido que haya sido obtenido y configurado a través de la coalición de K certificados parciales. El proceso de asociación de una dirección IP se da de la siguiente forma:




Asociación de una dirección IP a un nuevo nodo.

Donde:

1. El cliente envía un mensaje broadcast addr_req.
2. Al recibir una petición, el nodo servidor responde enviando un mensaje addr_rep. Es posible que dos o más nodos servidores respondan al mensaje de petición.
3. El nodo cliente selecciona sólo el nodo servidor con mayor cantidad de direcciones IP libres y envía de vuelta al nodo servidor un mensaje server_poll, ignorando la respuesta de los otros nodos servidores.
4. Al recibir el mensaje server_poll, el nodo servidor está listo para asociar una dirección IP al nodo cliente. Divide su free_ip_block a la mitad, enviando la mitad para el nodo cliente y guardando la otra mitad.
5. Cuando el nodo cliente recibe el mensaje ip_assigned, aloja toda la estructura en la tabla local free_ip_block, donde la primera dirección IP es usada para la configuración de su interfaz y las restantes direcciones IP se utilizan para servir a otros nodos. Después de esto, el nodo cliente envía un mensaje ip_assignment_ok hacia el nodo servidor indicando que la configuración fue realizada con éxito. Al adquirir correctamente la dirección IP, termina el proceso de autoconfiguración.

Si el nodo servidor no recibe el último mensaje del nodo, envía un “ping” al nodo cliente para verificar si fue definitivamente configurado. Si recibe alguna respuesta, implica que el proceso de autoconfiguración fue realizado con éxito.

Al recibir el free_ip_block que contiene su dirección IP, el nodo cliente y el nodo servidor pasan a ser llamados “nodos amigos”. Esta connotación dada a los nodos facilita el proceso de recuperación de direcciones IP cuando hay fallo en algún nodo o cuando ocurre la pérdida de algún mensaje. Así, el nodo servidor mantiene una tabla local (buddy_ip_blocks) con el identificador de todos los nodos para los que es asoció una dirección IP. En la tabla 4.1, tenemos un ejemplo de una tabla conteniendo tres nodos asociados por un nodo servidor que inicialmente tenía la free_ip_block (192.168.67.0 – 192.168.67.255) y, después de la distribución de direcciones IP a los nodos clientes, se quedó con el free_ip_block (192.168.67.0-192.168.67.127). Usamos notación de direcciones IP para mostrar la estructura de la tabla.

Tabla de un nodo servidor con las direcciones IP de los nodos amigos.

| **Identificación de un nodo amigo** | **Direcciones IP libres de un nodo amigo** |
|---|---|
| 192.168.0.128 | 192.168.0.129 – 192.168.0.254 |
| 192.168.0.64 | 192.168.0.65 – 192.168.0.127 |
| 192.168.0.32 | 192.168.0.33 – 192.168.0.63 |

Es posible que un nodo servidor no tenga ninguna dirección IP disponible en su free_ip_block al recibir un mensaje de petición de dirección IP. La solución es encaminar la petición a los nodos vecinos en la red. Para mantener una distribución uniforme de las direcciones IP, el nodo servidor investiga en su tabla de direcciones IP asociados a los nodos amigos por el nodo que tenga el mayor free_ip_block. El nodo servidor envía una petición de una dirección IP para este nodo amigo. El nodo amigo divide su free_ip_block en dos partes iguales y suministra la mitad al nodo servidor que traspasa para el nodo cliente. La elección del nodo amigo con el mayor free_ip_block posee la ventaja de no necesitar el intercambio de mensajes adiccionales, apenas una investigación en la tabla de direcciones de nodos amigos.

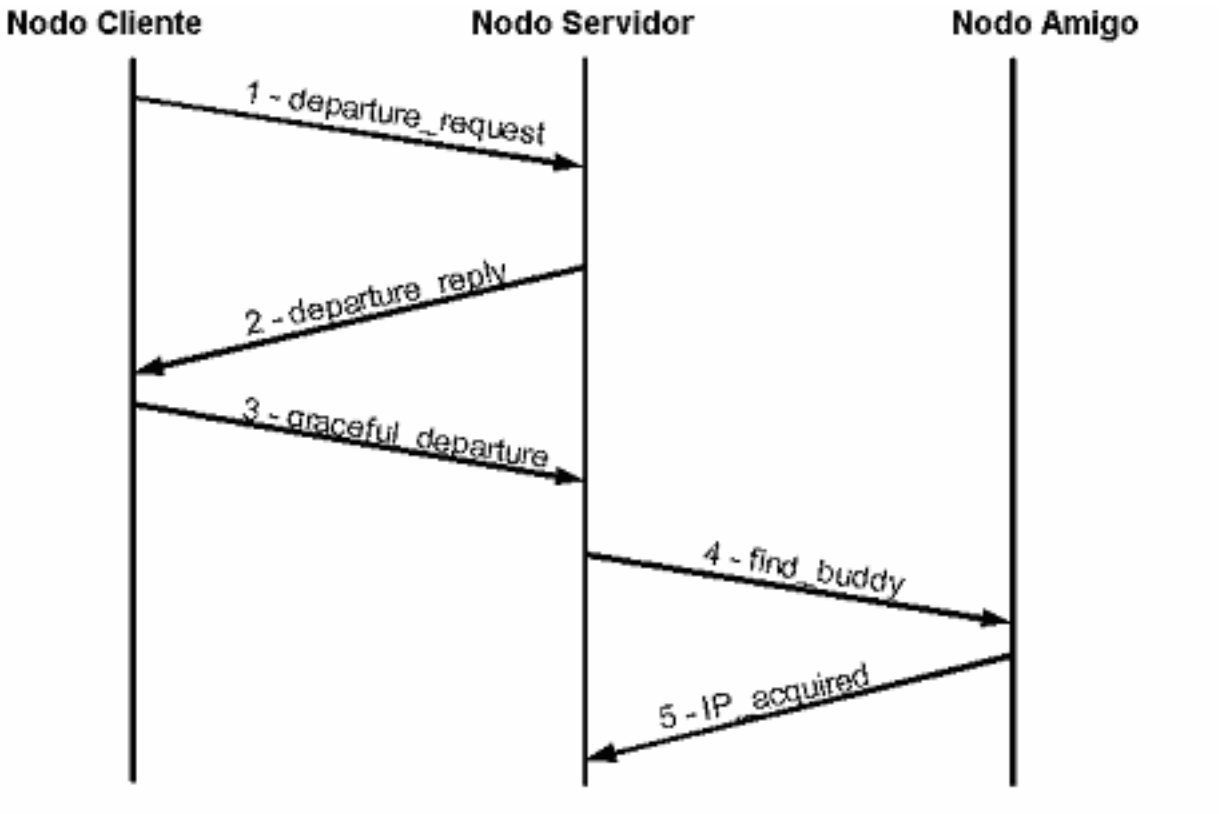
Si ninguno de los nodos amigos del nodo servidor posee una dirección IP libre, el nodo servidor responde al nodo cliente, enviando un mensaje deny diciendo que no hay direcciones IP disponibles en ese momento.

### 3.2.3 *Salida de Nodos*

Debido a las constantes variaciones en la topología de la red como consecuencia de la movilidad de los nodos, habrá situaciones en que los nodos saldrán de la franja de comunicación de sus vecinos, este proceso será tratado como si el nodo hubiese dejado la red ad hoc. El proceso de salida o fallo de un nodo debe ser cuidadosamente analizado con el fin de evitar problema de distribución de direcciones IP para nuevos nodos. El protocolo es proyectado para minimizar el trabajo del nodo cliente, permitiendo que el mismo deje la red e intercambie el menor número de mensajes posibles con sus nodos vecinos. Eso es porque el nodo que desea salir de la red no tiene tiempo o energía suficiente para realizar el intercambio de mensajes con sus nodos vecinos.

Por tanto, habrá dos situaciones en que se considera que un nodo deje la red. La primera situación, cuando el nodo desea salir de la red y avisa a sus vecinos de su intención. La segunda, detectar la salida de un nodo cuando ocurre un fallo en el nodo o el nodo deja la red sin avisar a sus vecinos.

Salida fácil de nodos




Proceso de salida fácil de un nodo.

Donde:

1. El nodo que desea dejar la red envía un mensaje en difusión departure_request.
2. Al recibir el mensaje departure_request, un nodo vecino, que colabora con un nodo servidor, responde la petición enviando un mensaje departure_reply al nodo cliente. El nodo cliente puede recibir varios mensajes departure_reply, aunque sólo aceptará el primer mensaje de respuesta e ignorará los demás. Así, cuando el nodo cliente recibe el mensaje tiene la certeza de que a partir de ese momento, el nodo servidor se hace responsable de su free_ip_block.
3. El mensaje graceful_departure sirve para confirmar el deseo del nodo cliente de dejar la red. Este mensaje es el que contiene definitivamente el free_ip_block del nodo cliente.
4. Al adquirir el free_ip_block y obtener la confirmación de intención del nodo cliente de dejar la red, el nodo servidor debe encontrar el nodo amigo del nodo cliente y enviar el free_ip_block adquirido, hacia el nodo amigo a través del mensaje find_buddy. La devolución de un

free_ip_block hacia un nodo amigo es importante para el mantenimiento de la consistencia de la tabla local de direcciones IP asociadas en cada nodo. Si el nodo amigo del nodo que está dejando la red no recibe el mensaje find_buddy, o esta fuera del alcance de transmisión del nodo servidor, las direcciones libres son agregados a la free_ip_block del nodo servidor.

5. Así, cuando el nodo servidor recibe el mensaje ip_acquired, se asegura de que el free_ip_block fue adquirido con éxito por el nodo amigo, terminando así el proceso de salida fácil de un nodo.

Acordando que la pérdida de cualquiera de los mensajes citados arriba, implica que el nodo deja la red sin avisar.

**Salida brusca o fallo de nodos**

Habrá un momento en la red en que ningún nodo tendrá direcciones IP libres en su free_ip_block. Eso necesariamente no significa que todas las direcciones IP estén en uso. Es posible que algún nodo haya fallado (dejado bruscamente la red, batería agotada o pérdida de mensajes durante la transmisión) y no haya enviado el mensaje departure_request hacia los nodos vecinos llevando consigo las direcciones IP que estaban en su free_ip_block. Asumiendo este escenario, es necesaria la recolección de las direcciones por parte de otros nodos.

Con base en la tabla de direcciones IP asociadas a los nodos amigos, un nodo puede detectar si el nodo amigo ha salido de la red abruptamente. Se suponen dos nodos amigos, el nodo Alice y el nodo Bob. Para detectar el fallo de dirección IP, el nodo Alice irá a buscar por la dirección IP que suministró al nodo Bob y vice-versa. El nodo Alice intenta entonces verificar si el nodo Bob está activo enviando un mensaje “ping”. Si el nodo Alice recibe alguna respuesta de el nodo Bob, significa que el nodo Bob todavía está en la red, en caso contrario, Bob ha salido de la red y el nodo Alice asume que dejó la red abruptamente y realiza la recuperación de direcciones IP sumando el free_ip_block del nodo Bob a su free_ip_block.

Este método de recuperación de direcciones posee sus ventajas y desventajas. La gran ventaja es que cada nodo es responsable de encontrar sus nodos amigos y no hay necesidad de hacer un broadcast en la red. La desventaja de este método es que si el nodo amigo se ha movido del rango de transmisión de la red temporalmente y vuelve en seguida, se considerará que dejó la red. Por consiguiente, el nodo amigo responsable de la detección del posible fallo del nodo, hace la recuperación de direcciones IP. Por fin, el nodo que sale temporalmente de la red debe realizar nuevamente el proceso de autoconfiguración para la obtención de una dirección IP.

En este momento, el nodo que sale de la red temporalmente aún posee una dirección IP, pero esa dirección IP también forma parte de la free_ip_block del nodo servidor, de manera que, cuando el nodo vuelva a la red, él debe solicitar una nueva dirección IP, ya que la que tiene no es válida.

En la sección siguiente, se expone el proceso de sincronización de los nodos, de gran importancia en el proceso de detección de fallos de nodos y en la recuperación de direcciones IP.

### 3.2.4 *Proceso de Sincronización*

En la asignación de direcciones IP a un nuevo nodo de la red, fue necesaria la creación de una tabla que muestra la distribución de direcciones IP hechas a los nodos amigos (buddy_ip_blocks). Cada nodo perteneciente a la red posee esta tabla. En el proceso de sincronización de la red, interviene el uso de esta tabla creada anteriormente. Con las informaciones de cada tabla almacenada localmente en cada nodo, se monta otra tabla en la que se almacenan las direcciones asignadas para todos los nodos de la red manteniendo la actual situación de la topología de la red. Por tanto, en esta tabla global llamada allocated_ip_blocks, se tienen todos los nodos de la red y sus respectivos conjuntos de direcciones IP.

Los nodos de una red ad hoc sincronizan la tabla global de direcciones IP asignadas (allocated_ip_blocks) en intervalos de tiempo para detectar posibles fallos en el direccionamiento. La sincronización se realiza de la siguiente: cada nodo envía un mensaje broadcast con su tabla de direcciones IP asociadas (buddy_ip_blocks), con la intención de actualizar las tablas globales (allocated_ip_blocks) de todos los nodos vecinos.

A cada entrada o salida de un nodo en la red, la tabla local (buddy_ip_blocks) es actualizada. En otras palabras, cada nodo con una entrada nueva en la tabla significa que hay un nodo nuevo en la red o la divisón de un free_ip_block en bloques menores. De la misma manera, excluir una entrada de la tabla significa la salida de algún nodo o la unión de dos free_ip_block en un único bloque. Cuando un nodo actualiza su tabla localmente, no informa inmediatamente a todos los nodos de la red. Los otros nodos conocerán el cambio, solamente cuando ocurra el intervalo designado para que los nodos sincronicen sus tablas allocated_ip_blocks, a fin de mantener las informaciones de red actualizadas.

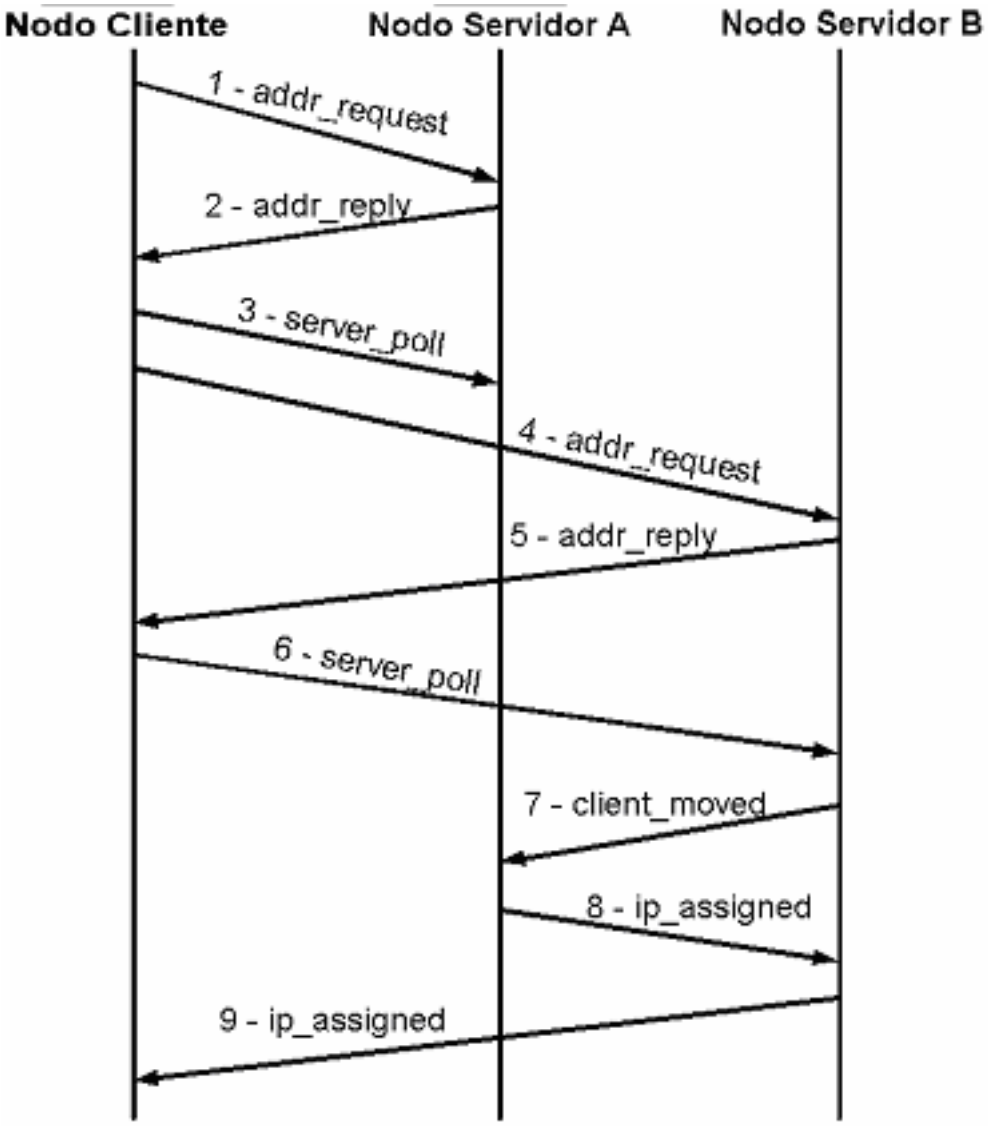
El tiempo de sincronización entre los nodos se establecerá en la configuración de un reloj global con respecto al cual actualizaran sus tablas evitando así la inconsistencia en el proceso de sincronización. Esta inconsistencia ocurre cuando una tabla antigua llega a los nodos vecinos después de una tabla más actual, debido al tráfico o al comportamiento de la red.

Así que un nodo recibe un mensaje de sincronización Update_net_info, compara su tabla con la tabla recibida y realiza las debidas alteraciones para adecuarse a la topología actual de la red.

### 3.2.5 *Migración de un nodo cliente*

Debido a la constante movilidad de los nodos, es posible que el nodo cliente se mueva durante el proceso de asociación de direcciones IP y con el movimiento pueda perder el contacto o estar fuera del rango de transmisión del

nodo servidor, el cual estaba le asociando una dirección IP. En este caso, el nodo cliente escoge un nuevo nodo servidor e informa al mismo de la existencia de otro nodo servidor. El nuevo nodo informa al otro nodo de la migración del nodo cliente. Cuando el proceso de asociación de direcciones IP estaba terminado en el otro nodo servidor, él encamina el resultado (free_ip_block) para el nuevo nodo servidor que encamina para el nodo cliente.




Migración de un nodo cliente.

Donde:

1. El nodo cliente envía un mensaje broadcast addr_req pidiendo una dirección IP.
2. Al recibir la petición, el nodo servidor responde enviando un mensaje addr_rep. Es posible que dos o más nodos servidores respondan al mensaje.
3. El nodo cliente sólo selecciona el nodo servidor con la mayor cantidad de direcciones IP libres en el free_ip_block y envia de vuelta al nodo

servidor un mensaje server_poll, ignorando la respuesta de los otros nodos servidores.

4. En el momento que el nodo cliente se mueve y está fuera del alcance del nodo servidor A. Así el envía nuevamente un mensaje addr_req avisando de su primer proceso de configuración con el nodo servidor A.
5. Entonces, los nodos servidores propagan el mensaje addr_rep para que el nodo cliente pueda escoger el vecino más próximo y que este dentro del rango de transmisión del nodo cliente A.
6. El nodo cliente escoge el nodo servidor B para hacer el proceso de autoconfiguración.
7. El nodo servidor B envía un mensaje client_moved avisando al nodo servidor A que el nodo cliente se movió y que ahora el es el nuevo servidor cliente.
8. El nodo servidor A entonces encamina el resultado de la asociación de la dirección IP por el nodo servidor B.
9. El nodo servidor B pasa el resultado al nodo cliente configurándolo correctamente.

En la ilustración anterior, el nodo servidor A también se puede mover y interrumpir el proceso de autoconfiguración. El tratamiento de esta característica es idéntico al hecho con el movimiento del nodo cliente. Para resolver este problema, basta que el nodo cliente informe al nodo servidor B que el proceso de autoconfiguración fue iniciado por el nodo servidor A. Por fin, el nodo servidor B se comunica con el nodo servidor A finalizando todo el proceso de configuración del nodo cliente.

El proceso de migración del nodo cliente en el proceso de asignación de direcciones IP requiere el intercambio de muchos mensajes y de la colaboración de dos nodos para que sea ejecutado con éxito. Por eso, existe la necesidad de que las tablas de los nodos estén actualizadas con la topología actual de la red.

### 3.2.6 *Perdida de mensajes*

El protocolo de autoconfiguración utiliza mensajes para los procesos de asociación salida y sincronización, se utilizan mensajes UDP/IP al ser un protocolo no orientado a conexión es muy importante controlar la perdida de paquetes para evitar que existan nodos con IP duplicada.

Para tratar la pérdida de mensajes, evitando problemas en el funcionamiento del protocolo, se usan mensajes de confirmación y temporizadores apropiados para el cambio de mensajes, garantizando así que cada servicio sea realizado dentro de un tiempo finito. En la sección siguiente se describe cada tiempo utilizado en la comunicación entre los nodos.

Un nodo desea obtener una dirección IP enviando un mensaje addr_req. Siguiendo el proceso de reconfiguración, los nodos servidores responden a la solicitud. Entonces, el nodo cliente escoge nodo servidor que por su parte

asocia una dirección IP y suministra un free_ip_block. Por fin el nodo cliente envía el último mensaje confirmando la autoconfiguración. Suponiendo que el último mensaje (ip_assignment_ok) se pierde y no llegue hasta el nodo servidor confirmando la asociación de la dirección IP al nodo cliente. En este momento, el nodo servidor ya hizo el envío del free_ip_block. Sin embargo, como el mensaje de confirmación no llegó, el nodo servidor no sabe si fue su mensaje ip_assigned que se perdió por el camino o fue el mensaje del nodo cliente que no llegó de vuelta. Cuando el nodo cliente tiene una dirección IP asociada, posee un free_ip_block, pero no consta en la tabla buddy_ip_blocks del nodo servidor. La pérdida de ese mensaje descrito en el escenario anteriormente puede acarrear en un mal funcionamiento del protocolo, lo que es resuelto con la especificación del temporizador para cada mensaje intercambiado en el protocolo.

### 3.2.7 *Detección de unión y separación de redes*

El constante movimiento de los nodos puede llevar la división de una red ad-hoc en dos o más redes. Esas redes pueden converger y hacerse una única red nuevamente. En esa posible unión de redes, puede ocurrir que dos nodos posean la misma dirección IP, por lo tanto el gran desafío aquí es detectar cuando ocurre la partición y como tratar la unión de dos redes llevando a un posible conflicto de direcciones IP. Una gran ventaja del protocolo de autoconfiguración especificado, es que reacciona bien ante la partición y la unión de redes Manet.

En este protocolo, cada partición o cada red ad-hoc posee una identificación única que ya fue descrita anteriormente, llamada partition_id y que posee el objetivo de ayudar en la detección de la unión de dos o más redes. El partition_id es determinado en la inicialización de la red, así, su valor es derivado del certificado emitido por el nodo líder que hace la distribución de las llaves parciales para los primeros K nodos de la red. Por lo tanto, para diferenciar dos o más redes, cada partición es asociada con un único y exclusivo partition_id.

Para resolver esos problemas y evitar direcciones duplicadas en la red, cuando se produce la unión de dos o más redes, las siguientes soluciones son propuestas:

**Detectando la partición de una red ad-hoc**

Considerando que cada partición deba poseer su identificador único, considere que una red ad-hoc se divida en dos redes diferentes. En ese momento, se puede comparar la división de la red ad-hoc como si fuera una salida abrupta de varios nodos, donde los nodos dejaron la red sin avisar y llevaron una determinada cantidad de direcciones IP.
De esa forma, como fue descrito anteriormente, los nodos envían un mensaje “ping” para sus nodos amigos y en caso de que no respondan, el nodo considera que su nodo amigo salió de la red, recuperando sus direcciones IP. Ese proceso es gradual y, al poco tiempo, los bloques de direcciones IP van

siendo recuperados. En otras palabras, la detección de la separacón de una red ad-hoc no es más que una salida de un grupo de nodos al mismo tiempo.

**Unión de dos o más redes**

La detección de una unión de dos o más redes Manet se da a través de un proceso muy simple. Aprovechando que los nodos intercambian mensajes Hello periódicamente, es posible detectar de forma clara y eficiente cuando dos o más redes se juntaron. Esos mensajes Hello contienen la dirección IP y el partition_id del nodo. Cuando un nodo recibe un mensaje Hello contiendo un partition_id diferente de la suya, él detecta que hubo una unión de dos redes ad-hoc. Por fin, el nodo con mayor dirección IP envía un mensaje de broadcast para recolectar las configuraciones actuales de la red, para saber las direcciones asociadas a los nodos de la red.
La elección del nodo que hace la colecta de las configuraciones de la red es de elección del administrador de la red ad-hoc y puede ser el nodo que posee la mayor dirección IP tanto como el nodo que posee la menor dirección IP.

Para que los nodos de diferentes redes puedan comunicarse y formar una nueva red ad-hoc, es necesario cumplir el requerimiento principal que es el establecimiento de una relación de confianza cruzada entre las autoridades certificadoras distribuidas de cada red.

Cuando se produce la unión de dos redes ad-hoc, la cantidad de nodos puede superar las 256 direcciones IP disponibles del direccionamiento. En ese caso, la solución a tomar es el cambio de la clase de direccionamiento de la red IP que en ese ejemplo, sería el cambio del direccionamiento utilizando la clase C que comporta a lo sumo 256 nodos para el direccionamiento utilizando la clase B que soporta redes con hasta 65536 nodos.

## 3.3 Mensajes y Estuctura de Datos del Protocolo

El protocolo de autoconfiguración es eficiente por poseer un número finito de mensajes para cada operación. Todas las operaciones descritas y especificadas en el protocolo como, la asociación de una dirección IP, la salida de un nodo, la sincronización de la red y la unión y la separación de redes poseen un tiempo finito para el cambio de mensajes garantizando así un óptimo funcionamiento y una baja latencia.
Se describen a continuación todas las estructuras de datos y tablas que son utilizadas en el cambio de mensajes del protocolo.

### 3.3.1 *Estructura de datos y tablas utilizadas*

Las siguientes estructuras de datos son necesarias para mantener actualizadas las informaciones de la red. Cada tabla especificada debe ser almacenada localmente en todos los nodos. Las tablas son actualizadas en intervalos de tiempo determinados en todos los nodos como descrito en el proceso de sincronización. Considerando un nodo i, tenemos:

**My_IP_blocki**: estructura que almacena en que bloque está la dirección IP del nodo. La dirección IP del nodo es la primera dirección del bloque.

**Free_IP_blocksi:** tabla que contiene las franjas de direcciones IP libres para atender los nuevos nodos, es decir, son las direcciones que están libres para que sean asociados a los nuevos nodos que llegan a la red. Para esa tabla, se tiene una característica importante: FreeIPblocks <> *FreeIPblocks j.

**Buddy_IP_blocksi**: tabla que contiene el FreeIPBlock de cada nodo amigo. Esa tabla es semejante a la Tabla 4.8 y es muy útil en la recuperación de las direcciones IP de nodos que partieron abruptamente de la red. Aquí, también vale la misma característica citada arriba: BuddyIPblocksi <> BuddyIPblocksj.

**Allocated_IP_blocksi:** tabla que contiene todas las direcciones asociadas en la red así como todos sus respectivos freeIPblock. Esa tabla está siempre actualizada, pues tiene el objetivo de representar la topología más actual de la red.

Para cada red ad-hoc, tenemos un identificador único (partition_id) que representa la cual partición el nodo pertenencia. El primer nodo de la red es el responsable por la generación del partition_id utilizando su dirección de hardware y su certificado.

### 3.3.2 *Formato de los mensajes*

El formato de cada mensaje utilizado en el protocolo seguro de autoconfiguración, acordando que la extensión de autenticación está presente en todos los mensajes intercambiados para garantizar la seguridad del protocolo, se detalla a continuación

Mensajes del protocolo de autoconfiguracion.

| Mensaje | Descripción |
|---|---|
| ADDR_REQ | Mensaje que el nodo cliente envía al servidor para solicitar una IP. |
| ADDR_REP | Respuesta del servidor al nodo cliente cuando recibe el mensaje de solicitud de IP. |
| SERVER_POLL | El nodo cliente envía ese mensaje para un nodo servidor diciendo que lo escogió para hacer la configuración de su dirección IP. |
| IP_ASSIGNED | Mensaje enviado por el nodo servidor al nodo cliente asociando la dirección IP. |
| IP_ASSIGNMENT_OK | Mensaje enviado por el nodo cliente al nodo servidor confirmando que el proceso de autoconfiguración fue realizado con éxito. |
| DEPARTURE_REQ | Enviado por el nodo cliente a todos los nodos vecinos informando su intención de dejar la red. |
| DEPARTURE_REP | Mensaje que el nodo servidor envía al nodo cliente para que el mismo confirme el antojo de salir de la red. |
| GRACEFULL_DEPARTURE | Enviada por el nodo cliente para el nodo servidor indicando que él está definitivamente dejando la red. |

**ADDR_REQ**

Mensaje que el nodo cliente envía para la obtención de una dirección IP. La dirección origen es la dirección MAC de su interfaz de red y la dirección destino es la dirección de broadcast.

| 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Type | | | | | | | | Res | | | | | | | | Lenght | | | | | | | | | | | | | | | |
| Client MAC Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Client MAC Address | | | | | | | | | | | | | | | | Aligment | | | | | | | | | | | | | | | |

El formato del mensaje addr_req está ilustrado arriba y contiene 12 bytes con los siguientes campos:

Campos del mensaje *addr_req*.

| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | Addr_req |
| Length | 16 | Enviado como cero e ignorado en la recepción |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepción |
| Client Mac Address | 48 | Direccion de hardware do nodo cliente |
| Alignment | 16 | Enviado como 0 e ignorado en la recepción. |

Tiempo: el nodo cliente inicia el tiempo después de enviar el mensaje de requisito y espera un mensaje de respuesta de algún nodo servidor de la red. Cuando el tiempo expira y ningún mensaje es recibido de los nodos vecinos, él envía un nuevo mensaje de requisito por T veces, donde T es la cantidad máxima de tentativas de obtener una dirección IP.

**ADDR_REP**

Mensaje que el nodo servidor envía para el nodo cliente cuando él recibe el mensaje addr_req. El nodo cliente puede recibir este mensaje de varios nodos servidores, cosechando las informaciones de cada respuesta, sin embargo, él escoge el nodo servidor con el mayor free_ip_block.

| 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Type | | | | | | | | Res | | | | | | | | Lenght | | | | | | | | | | | | | | | |
| Server MAC Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| Server MAC Address | Aligment |
|---|---|
| IP_Block (IP) | |
| IP_Block (cicdr mask) | |

El formato del mensaje addr_rep está ilustrado arriba y contiene 20 bytes con los siguientes campos:

Campos del mensaje *addr_rep*.

| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | Addr_rep |
| Length | 16 | Enviado como cero e ignorado en la recepción |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepción |
| Server Mac Address | 48 | Direccion de hardware do nodo servidor |
| Alignment | 16 | Enviado como 0 e ignorado en la recepción. |
| IP_Block | 64 | IP address + CIDR mask |

## SERVER_POLL

Mensaje del nodo cliente enviada únicamente para el nodo servidor diciendo que lo escogió entre todos los mensajes addr_rep recibidas por contener el mayor free_ip_block. Ese mensaje contiene sólo la dirección de hardware del nodo cliente. Después del recibimiento de ese mensaje por el nodo servidor, comienza el proceso de autoconfiguración del nodo cliente.

| 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Type | | | | | | | | Res | | | | | | | | Lenght | | | | | | | | | | | | | | | |
| Client MAC Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Client MAC Address | | | | | | | | | | | | | | | | Aligment | | | | | | | | | | | | | | | |

El formato del mensaje server_poll está ilustrado arriba y contiene 12 bytes con los siguientes campos:

Campos del mensaje *server_poll*.

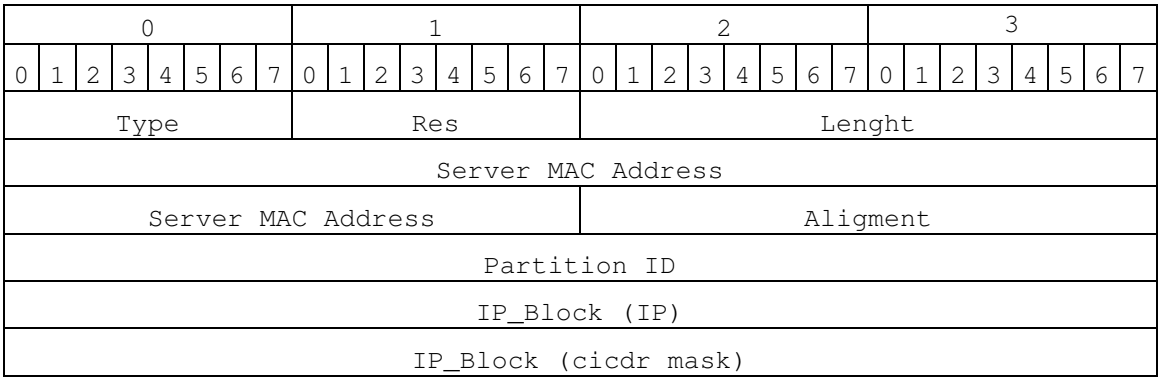
| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | Server_poll |
| Length | 16 | Enviado como cero e ignorado en la recepcion |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepcion |

| | | |
|---|---|---|
| Client Mac Address | 48 | Direccion de hardware do nodo cliente |
| Alignment | 16 | Enviado como 0 e ignorado en la recepcion. |

Timer: el nodo cliente inicia ese tiempo después del envío del mensaje server_poll y espera un mensaje ip_assigned del nodo servidor. Cuando el tiempo expira, el nodo cliente asume que el nodo servidor dejó la red antes de completar el proceso de autoconfiguración enviando un nuevo mensaje server poll por T veces, donde T es la cantidad máxima de tentativas de comunicar con el servidor escogido para suministrar el *free_ip_block.

### *IP_ASSIGNED*

Mensaje enviado por el nodo servidor al nodo cliente asociando la dirección IP al nodo cliente. Conjuntamente con ese mensaje sigue el identificador único de la partición, el partition_id. Por lo tanto, este mensaje contiene las siguientes informaciones: {PID, free_ip_block}. La primera dirección del free_ip_block es asociado a la interfaz del nodo cliente y el restante de las direcciones se queda almacenado en la estructura free_ip_block del nodo cliente y sirve para que él pueda atender el requisito de otros nodos, haciéndose un nodo servidor.




El formato del mensaje ip_assigned está ilustrado arriba y contiene 24 bytes con los siguientes campos:

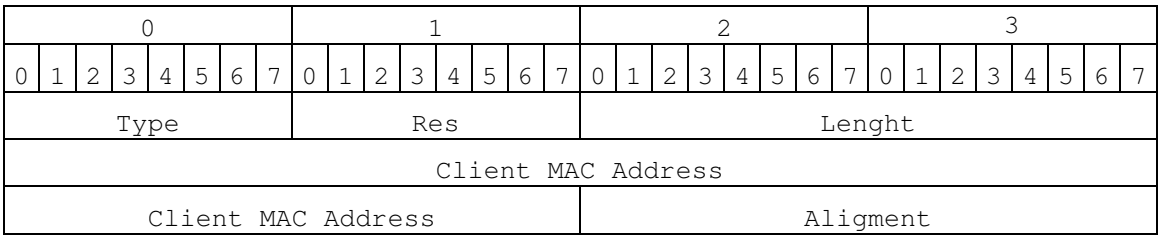
Campos del mensaje ip_assigned.

| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | IP_assigned |
| Length | 16 | Enviado como cero e ignorado en la recepcion |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepcion |
| Server Mac Address | 48 | Direccion de hardware do nodo servidor |
| Alignment | 16 | Enviado como 0 e ignorado en la recepcion |
| Partition_id | 32 | Identificador único da particion |
| IP_Block | 64 | IP address + CIDR mask |

Timer: el nodo servidor inicia ese tiempo después de enviar un mensaje ip_assigned y espera el mensaje ip_assignment_ok confirmando que el proceso de autoconfiguración fue concluida con éxito.

**IP_ASSIGNMENT_OK**

Mensaje del nodo cliente enviada únicamente para el nodo servidor afirmando que la dirección IP fue asociado con éxito. Después de ese mensaje, termina el proceso de autoconfiguración.




El formato del mensaje ip_assignment_ok está ilustrado arriba y contiene 12 bytes con los siguientes campos:

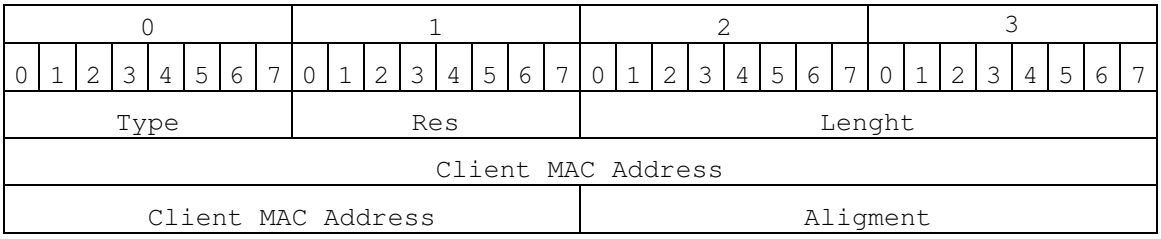
Campos del mensaje ip_assignment_ok.

| Campo | tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | IP_assignment_ok |
| Length | 16 | Enviado como cero e ignorado en la recepcion |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepcion |
| Client Mac Address | 48 | Direccion de hardware do nodo cliente |
| Alignment | 16 | Enviado como 0 e ignorado en la recepcion. |

Se nota que los mensajes intercambiados entre los nodos poseen una cabecera compuesta de un campo con el tipo del paquete, un campo que especifica el tamaño del paquete y por fin un campo reservado para uso futuro. Los mensajes poseen tamaño y tiempo limitados, garantizando la eficiencia del protocolo así como su robustez ante la pérdida de mensajes. Los próximos tres mensajes se refieren al proceso de la salida de un nodo de la red ad-hoc.

**DEPARTURE_REQ**

Mensaje en el nodo cliente enviada para todos sus vecinos informando su intención de dejar la red y consecuentemente liberar sus direcciones IP.



El formato del mensaje departure_req está ilustrado arriba y contiene 12 bytes con los siguientes campos:

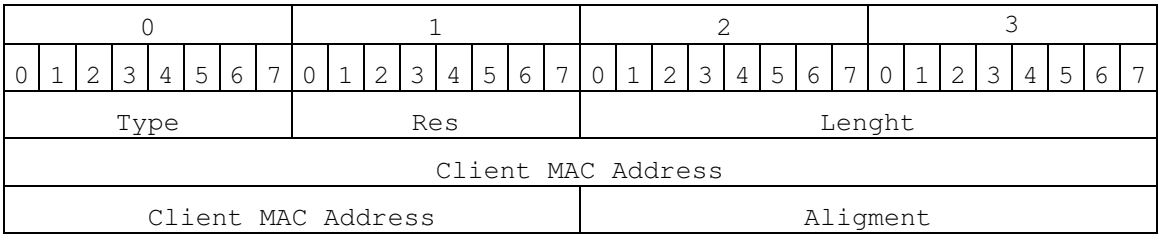
Campos del mensaje *departure_req*.

| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | Departure_req |
| Length | 16 | Enviado como cero e ignorado en la recepción |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepción |
| Client Mac Address | 48 | Direccion de hardware do nodo cliente |
| Alignment | 16 | Enviado como 0 e ignorado en la recepción. |

Timer: el nodo cliente inicia ese tiempo después del envío del mensaje departure_req y espera el mensaje departure_rep de un nodo servidor. Cuando el tiempo expira, el nodo cliente comienza el proceso de requisito de salida nuevamente o no envía cualquier mensaje y deja la red abruptamente, implicando en el proceso de recuperación de direcciones IP por los nodos amigos.

**DEPARTURE_REP**

Mensaje que el nodo servidor envía al nodo cliente para que el mismo confirme su intención de salir de la red. Después de ese mensaje, el nodo servidor se hace el responsable por las direcciones IP del nodo cliente. El nodo cliente acepta el primer mensaje de respuesta e ignora todas las otras.




El formato del mensaje departure_rep está ilustrado arriba y contiene 12 bytes con los siguientes campos:

Campos del mensaje departure_rep.

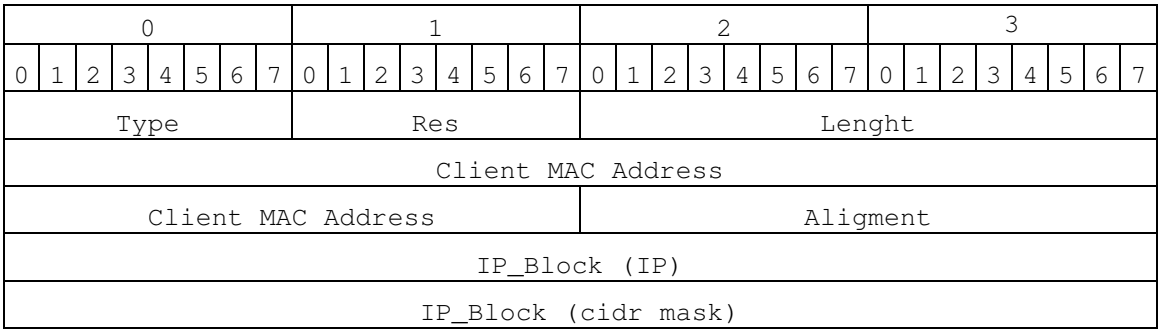
| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | departure_rep |
| Length | 16 | Enviado como cero e ignorado en la recepción |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepción |
| Client Mac Address | 48 | Direccion de hardware do nodo cliente |

| | | |
|---|---|---|
| Alignment | 16 | Enviado como 0 e ignorado en la recepción. |

Timer: el nodo cliente inicia ese tiempo después del envío del mensaje departure_rep y espera el mensaje graceful_departure del nodo cliente. Cuando el tiempo expira, el nodo servidor envía un ping al nodo cliente y cuando ya no hay respuesta, se asume que él dejó la red, siendo solamente ahora que el free_ip_block del nodo cliente es incorporado al free_ip_block del nodo servidor.

**GRACEFULL_DEPARTURE**

Mensaje que el nodo cliente envía para el nodo servidor indicando que él está definitivamente dejando la red. En ese momento, todas sus direcciones IP son




El formato del mensaje gracefull_departure está ilustrado arriba y contiene 20 bytes con los siguientes campos:

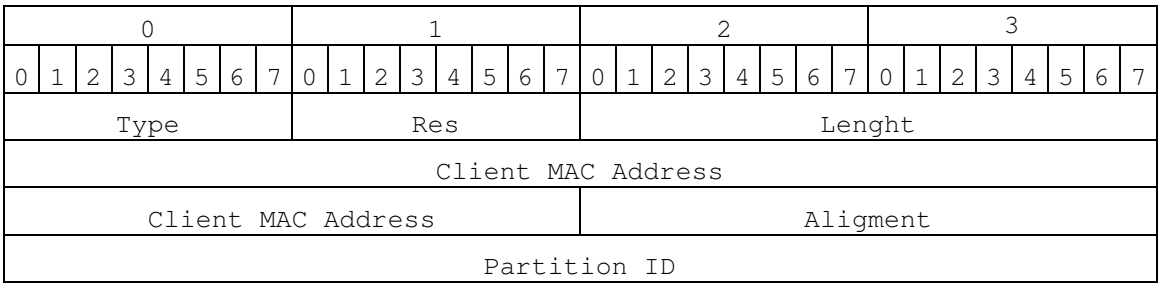
Campos del mensaje *server_poll*.

| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | Gracefull_departure |
| Length | 16 | Enviado como cero e ignorado en la recepcion |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepcion |
| Client Mac Address | 48 | Direccion de hardware do nodo cliente |
| Alignment | 16 | Enviado como 0 e ignorado en la recepcion. |
| IP_Block | 64 | IP address + CIDR mask |

A partir de ese momento, se asume que el nodo cliente dejó la red con éxito y todas las direcciones IP forman parte del free_ip_block del nodo servidor con el cual él estaba comunicándose. Así, no es necesario el proceso de recuperación de direcciones IP, que ocurre cuando un nodo deja la red abruptamente. Por fin, el nodo servidor ejecuta el comando “ping” para verificar que si el nodo cliente definitivamente dejó la red.

**HELLO**

Mensaje intercambiado entre todos los nodos contiendo sólo la identificación de la partición para que pueda ser detectada la unión de particiones, lo que ocurre cuando un nodo recibe un mensaje Hello con una identificación de partición diferente de la suya.




El formato del mensaje Hello está ilustrado arriba y contiene los siguientes campos:

Campos del mensaje hello.

| Campo | Tamaño (bits) | Valor |
|---|---|---|
| Type | 8 | IP_assigned |
| Length | 16 | Enviado como cero e ignorado en la recepcion |
| Res | 8 | Reservado - enviado como 0 e ignorado en la recepcion |
| Server Mac Address | 48 | Direccion de hardware do nodo servidor |
| Alignment | 16 | Enviado como 0 e ignorado na recepcion. |
| Partition ID | 32 | Identificador único da partição |

### 3.3.3 *Temporizadores utilizados*

La utilización de Temporizadores para el cambio de mensajes en el protocolo asegura que su funcionamiento sea correcto aún con eventuales pérdidas de mensajes o fallo de los nodos. La especificación de cada temporizador utilizado es configurada en la implementación, donde son hechas pruebas y simulaciones con temporizadores varios para evaluarse el comportamiento del protocolo.

La especificación de los temporizadores es hecha considerando varios factores externos como: la densidad de la red, la cantidad de nodos y la tasa de comunicación entre ellos. Así, cuanto mayor la cantidad de nodos, mayor debe ser el tiempo para que un mensaje llegue a su destino. Abajo, son descritos detalladamente cada temporizador utilizado en la asociación de una dirección IP a un nuevo nodo:

**Reply_timer:** el nodo cliente inicia ese temporizador después de enviar un mensaje de petición de dirección IP addr_req y espera el mensaje addr_rep del nodo servidor. Si el temporizador expira y no ha recibido cualquier mensaje, el nodo cliente envía nuevo mensaje de petición e inicia el temporizador. El nodo cliente repite el envío del mensaje addr_req por T veces donde T es la

cantidad máxima de tentativas que el nodo cliente hace para obtener una dirección IP.

**IP_assigned_timer**: el nodo cliente inicia ese temporizador después de enviar un mensaje Server_pool y espera el mensaje IP_assigned contiendo el free_ip_block. Si el temporizador expira y no ha recibido ningún mensaje, el nodo cliente asume que el nodo servidor dejó la red antes de completar el proceso de autoconfiguración. El nodo cliente repite el envío del mensaje Server_poll por T veces donde T es la cantidad máxima de tentativas de escoger el servidor para suministrar el free_ip_block.

**Confirm_timer**: el nodo servidor inicia ese temporizador después de enviar un mensaje ip_asigned a un nodo cliente y espera el mensaje ip_asigment_ok del nodo cliente confirmando que el proceso de autoconfiguración fue concluido con éxito. Si el temporizador expira y no ha recibido el mensaje de confirmación, el nodo servidor asume que el nodo cliente dejó la red antes de completar el proceso de autoconfiguración . En caso contrario, el nodo servidor actualiza su tabla de direcciones IP de los nodos amigos, incluyendo el nodo cliente.

Para la confirmación de la salida controlada de un nodo, son utilizados los siguientes temporizadores en el cambio de los mensajes entre los nodos:

**Depart_request_timer**: el nodo cliente inicia ese temporizador después de enviar un mensaje departure_request a un nodo servidor y espera el mensaje departure_reply. Si el temporizador expira y no ha recibido ningún mensaje, el nodo cliente posee dos opciones: o inicia nuevamente el proceso de salida de la red o deja la red asumiendo que ya no es parte de ella. En este último caso, la salida abrupta del nodo requiere que sea ejecutado el proceso de recuperación de las direcciones IP.

**Gracefull_departure_timer**: el nodo servidor inicia ese temporizador después de enviar un mensaje departure_reply a un nodo cliente y espera el mensaje graceful_departure. Si el temporizador expire y no haya recibido el mensaje, el nodo servidor repite el envío del mensaje departure_reply por T veces donde T es la cantidad máxima de tentativas del nodo servidor de recibir el mensaje graceful_departure del nodo cliente.

# 4 Arquitectura del Sistema

## 4.1 Protocolo de Enrutamiento

El protocolo de enrutamiento nos va a servir para poder realizar la comunicación correctamente entre los nodos que componen la red. Este protocolo tiene su base en el protocolo de enrutamiento RIP (Routing Information Protocol) en su versión 2 que apareció para mejorar la versión original del protocolo. A continuación veremos las principales características del protocolo para después dar una idea de los cambios realizados para la adaptación al entorno de las redes inalámbricas ad hoc o MANET.

El protocolo original de RIP esta definido en el RFC1058, mientras que el protocolo de enrutamiento para RIP versión 2 la especificación se encuentra en Internet Standard (STD) 56 o RFC 2453. De estos documentos es de donde hemos sacado toda la información del protocolo y que exponemos a continuación.

El protocolo utiliza un algoritmo basado en vector distancia y, en particular, en el algoritmo de Bellman-Ford; este algoritmo se basa en la búsqueda del camino óptimo mediante el conteo de saltos, considerando cada router atravesado para llegar al destino como un salto. La información para el correcto funcionamiento de este algoritmo se envia periódicamente entre los routers de la red, pero este punto se detallará más adelante. Gracias a este algoritmo este protocolo es adecuado para su utilización en IGP (Internet Gateway Protocol) en redes pequeñas, este IGP es un protocolo que utiliza las tablas de enrutamiento para la comunicación dentro de un sistema autónomo.

Entrando en el detalle de funcionamiento del protocolo podemos comentar que RIP mantiene una tabla con entradas para cada destino; estas entradas tienen como información importante la dirección del siguiente salto y la métrica. Periódicamente cada router envía la información de las rutas que conoce, indicando los destinos y las métricas. Cada router al recibir la información de sus vecinos suma el costo recibido con el costo que conlleva llegar al vecino y, si obtiene una distancia menor actualiza la ruta con los nuevos datos, y si es mayor desecha esa nueva ruta.

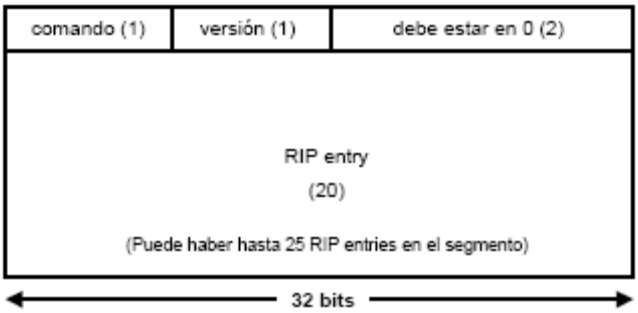
Por otro lado tenemos la métrica utilizada, que en RIP es el número de saltos hasta llegar al destino; para evitar el conteo hasta el infinito el número de saltos máximo se reduce a 16, siendo este número el que se utiliza para indicar el infinito o destino inalcanzable. Además para paliar el problema del conteo hasta el infinito se utiliza la técnica del horizonte dividido.

Ahora pasaremos a comentar el problema de las actualizaciones, si no se reciben actualizaciones de un ruta en un determinado tiempo se decide dar de baja esa ruta; dentro de las actualizaciones se pueden enviar métricas de 16 para indicar el destino ha sido declarada inaccesible, y si el router encuentra una ruta mejor envía una actualización de esa ruta para indicarla a todos los demás; en este protocolo los router sólo mantienen dentro de su tabla la ruta mejor en número de saltos, las demás son desechadas.

Dentro del protocolo RIP versión 1 teníamos la limitación de que no soportaba direcciones IP con máscaras variables, sólo era capaz de manejar máscaras de tipo A, B y C originales, y dentro de esta versión para cada destino se guardaba la siguiente información:

- La dirección IP del siguiente salto
- La métrica (número de saltos) para llegar a él.
- La dirección IP del próximo salto
- Banderas para indicar el estado de la actualización
- Temporizadores asociados a las entradas

Este protocolo utiliza UDP para la comunicación entre los enrutadores, y los mensajes son enviados por el puerto 520 de UDP. En cuanto al formato de los mensajes es el siguiente:




En este mensaje tenemos la capacidad para indicar el comando que queremos utilizar, que puede ser de tres tipos:

- Request(1): pedido del envió de la tabla de rutas. Normalmente se pedidos por broadcast cuando un router se levanta, pero también puede ser a un router en particular.
- Response(2): información de la tabla de rutas en respuesta a un request o enviadas por un update periódico.
- Traceon(3), traceoff(4) (obsoletos), reservados para SUN (5).

Dentro de campo versión deberemos introducir el valor '1' para indicar la versión 1 de RIP.

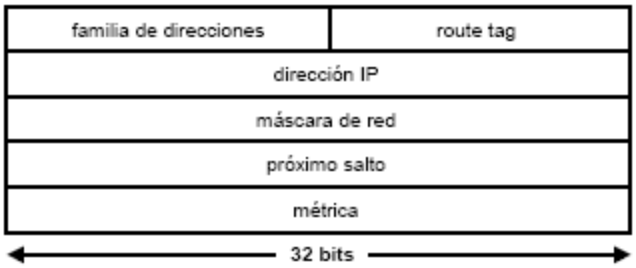
La siguiente característica que veremos es el formato de las entradas dentro de la tablas de rutas (RTE), que es el siguiente:

Donde tenemos que la familia de direcciones es igual a IP (2), la tabla de destinos es la dirección IP y la distancia es la métrica.

Un campo importante que tiene el protocolo RIP versión 2 es el campo temporizador; gracias a este campo se consigue la eliminación coherente de rutas debidos a falta de actualizaciones de estás. Por un lado tenemos que se generan actualizaciones de las rutas periódicamente, es decir, se generan mensajes “response” cada 30 segundos, además se evita la sincronización de actualizaciones con los demás routers gracias a un pequeño offset aleatoria que se integra a los 30 segundos. Por otro lado tenemos el tiempo de vencimiento de la ruta, en la cual si pasados 180 segundos no recibimos actualizaciones de la ruta lo hacemos es indicar esa ruta como no accesible, y para ello insertamos el valor 16 en la métrica; durante este tiempo si llega una actualización con la ruta lo que hacemos es actualizar la ruta con el valor correcto de la distancia. Una vez que tenemos un ruta en el esta obsoleto o no accesible, si transcurridos 120 segundos realizaremos una recolección de basura, en el cual borraremos de la tabla ese destino.

Ahora pasaremos a detallar el protocolo RIP versión 2, que extiende la funcionalidad de RIP versión 1. Esta actualización utiliza el mismo formato de mensajes que RIP versión 1 pero varía el formato de las entradas en la tabla de rutas (RTE) y agrega otros comandos (RIP entries). RIP versión 2 utiliza direcciones de multicast ( en vez de broadcast como en RIP versión 1) para los “requests”, la dirección multicast utilizada es 224.0.0.9 . RIP versión 2 permite la utilización de mensajes de RIP versión 1 si lo que recibe son este tipo de mensajes; esto es posible salvo que se explicite lo contrario.

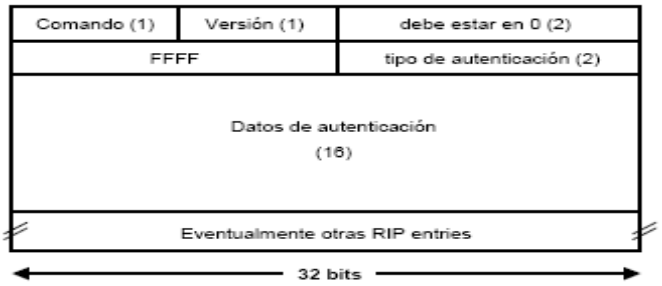
Como hemos dicho tenemos un formato distinto para las entradas en la tabla de rutas, este formato es el siguiente:




Donde el campo “route tag” es un atributo que se debe preservar y propagar para esa ruta. Se usa para separar rutas internas (IGP) de externas (EGP) y mejorar la interoperabilidad entre los protocolos. El campo próximo salto es capaz de optimizar si la IP es alcanzable por el receptor, sino se descarta.

Otro tema importante dentro de RIP versión 2 es la autenticación; esta autenticación se realiza por mensajes y sólo hay dos bytes sin uso en el formato del mensaje, entonces se usa el espacio de la primera rip entry para la autenticación y para indicarlo el campo “familia de direcciones” se pone a FFFF; debemos tener en cuenta que sólo debe haber una rip entry por mensaje

y está debe ser la primera. El formato del mensaje con autenticación es el siguiente:




Por último pasaremos a comentar la versión del protocolo RIP para IP versión 6, que es el llamado RIPng, este protcolo se define dentro del RFC 2080. Este protocolo solo debería implementarse en los enrutadores ya que los mecanismos de “router discovery” resuelven los problemas en los hosts.

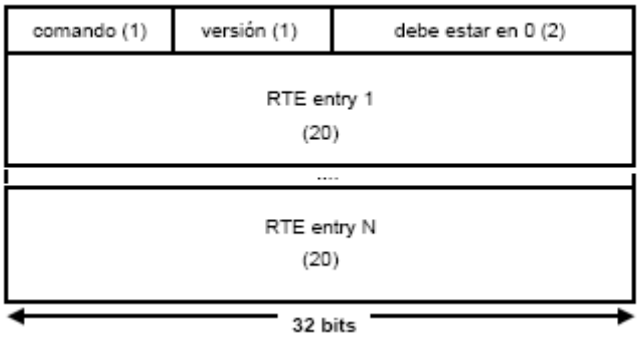
Cada entrada en RIPng deberá contener al menos la siguiente información :

- El prefijo IPv6 del destino
- La métrica (número de saltos) para llegar a ese destino
- La dirección IPv6 del próximo salto (next_hop).
- Banderas para indicar el estado de esa ruta (route change flag).
- Temporizadores asociados a la ruta.

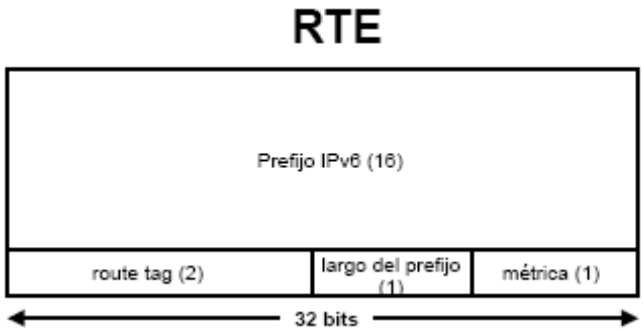
Podemos comprobar que en la base de esta nuevo protocolo RIP se encuentra tanto RIP versión 1 como versión 2, ya que no varía en demasía la información que debe contener este protocolo.

El formato del mensaje en RIPng es el mismo que en RIP versión 2, salvo que cambian las RTE (Route Table Entry, entradas en la tabla de rutas). Los formatos de los mensajes se pueden gráficamente en las siguiente figuras:

Formato para los mensajes de RIPng

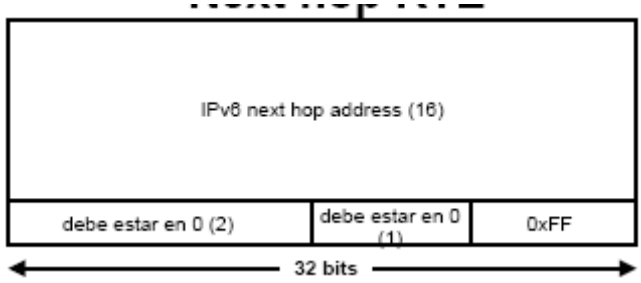



Formato para los mensajes de RTE




Donde el prefijo es un prefijo de IP versión 6 y tendrá 128 bits, el "route tag" es un atributa asignado y que debe preservarse y redistribuirse con la ruta, también permitirá las rutas de RIPng internas de las que son importadas de un EGP u otro IGP. El largo del prefijo será de 0 a 128, y la métrica se encontrará entre 1 y 16, donde el 16 será el infinito o no alcanzable.

Formato para el mensaje Next hop RTE




Este mensaje tiene la misma funcionalidad que en RIPv2 donde se sugiere el próximo salto, además es válido para todas las RTE que sigan a esta Next hop RTE, pero se corta con el fin del mensaje o cuando aparece otra Next hop RTE. El next hop address debe ser una dirección link-local. Si es 0 significa que la dirección es la de quién envía el mensaje.

Por otro lado los temporizadores en RIPng trabajan de la siguiente manera: cada 30 segundos más un tiempo aleatorio de más/menos 15 segundos se envía un mensaje de respuesta no solicitado con la información de todas las rutas a todos los vecinos; por otro lado el tiempo para dejar las entradas como no alcanzables es 180 segundos, y a partir de este tiempo se propaga la ruta como no alcanzable; por último el tiempo de borrado son 120 segundos más.

Por último para el tema de seguridad no se especifica ningún protocolo para RIPng, se confia en los mecanismo que posee IPv6 para obtener la seguridad requerida.

Resumiendo tenemos el protocolo RIPv2 para la implementación del protocolo de enrutamiento, pero debemos adecuar el protocolo a nuestro caso particular con redes inalámbricas, para ello debemos cambiar el tiempo de los temporizadores porque la estructura de una red ad hoc es cambiante y debemos intentar tener en cada momento una tabla de rutas fiable y coherente;

también cambiaremos la métrica utilizada para indicar el infinito para poder tener redes mayores a 16 nodos.

## 4.2 Módulo de Enrutamiento

Necesidades de establecer un protocolo de enrutamiento en la red ad-hoc:

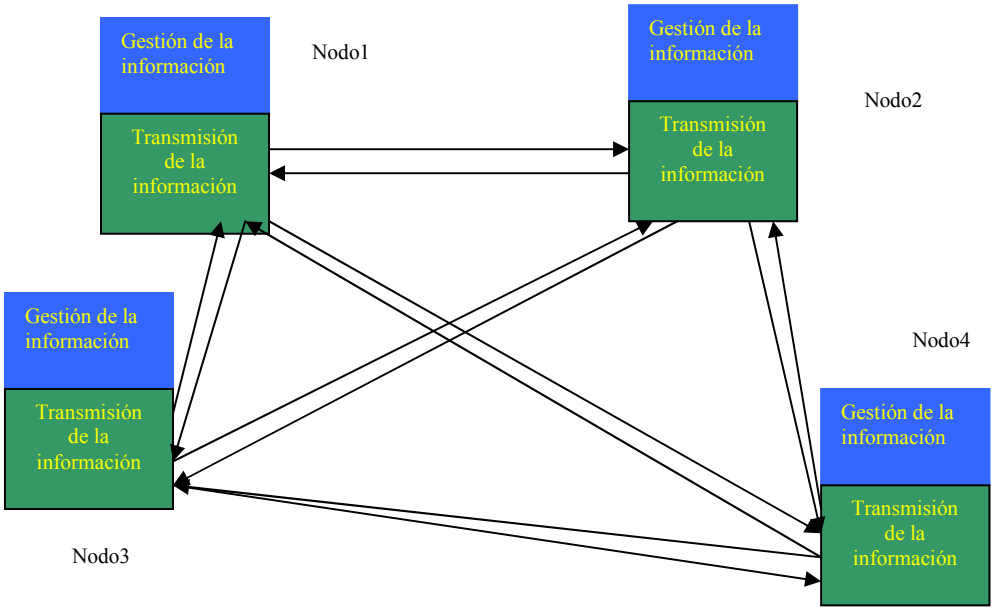
Como hemos descrito en apartados anteriores, para que la red ad-hoc tenga sentido y pueda dar servicios a los usuarios, necesitamos establecer un sistema de enrutamiento de tal manera que cada nodo de la red conozca el camino a seguir para llegar a cualquier otro, de esta manera la red ad-hoc ofrecería una capa de red que abstrae al usuario final de la implementación de la red, el funcionamiento será similar al de una red cableada, ya que a nivel de red es exactamente igual porque la diferencia reside en los niveles inferiores, este modulo debe estar integrado con los módulos de control de vecinos, modulo de autoconfiguración y con el modulo de sincronización de la red ad-hoc ya que cada modulo depende de los demás para su correcto funcionamiento.

A parte de las tareas realizadas que se detallan en puntos anteriores nuestro cometido en el proyecto ha sido la realización del modulo de enrutamiento, para proporcionar una capa de red a los sistemas conectados a la red ad-hoc como detallamos en el estudio de los protocolos de enrutamiento para redes ad-hoc hemos decidido implementar un protocolo similar a rip version 2, la modificación principal ha sido de los paquetes enviados que tienen un formato propio, en el cual ampliamos de 16 a 256 el numero de saltos posibles

El lenguaje de implementación a sido c UNIX, ya que el proyecto partía del modulo de autoconfiguración implementado en c y por uniformidad hemos continuado con este lenguaje de programación, se han utilizado las librerías libpcap para capturar paquetes, libnet para enviar paquetes, y las librerías de posix threads para mantener la concurrencia entre las operaciones de captura y envio de paquetes, mas adelante detallaremos cada una de estas librerías, la compilación ha sido realizada con el compilador gcc.

### 4.2.1 *Descripción general del funcionamiento de nuestro sistema*

Para conseguir realizar la tarea de enrutamiento, debemos tener en cuenta dos partes fundamentales la gestión de la información, y la transmisión de la información, también hay que mencionar el modulo de codificación/decodificación que veremos mas adelante.

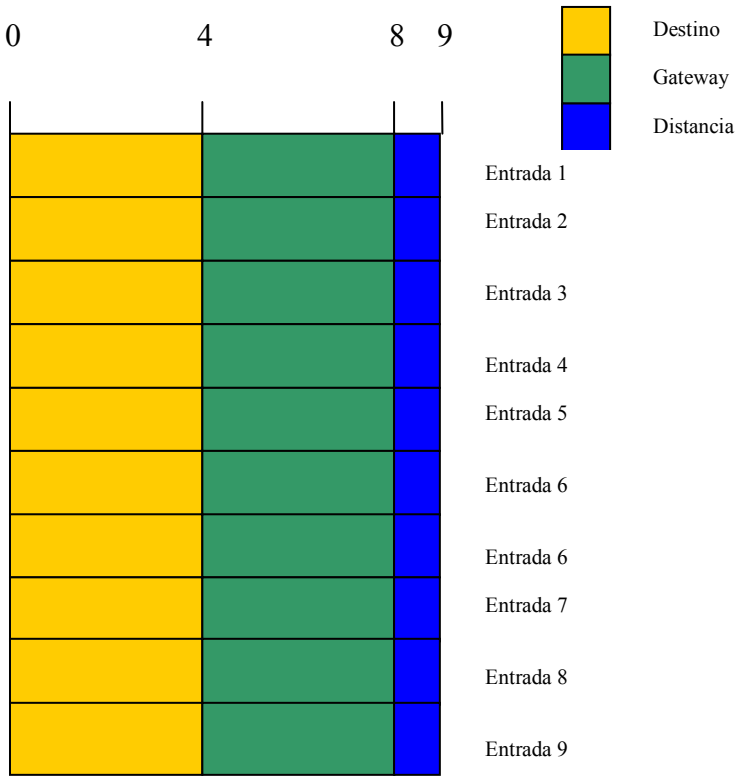



El modulo de gestión de la información, se encarga de interpretar la información recibida de los nodos vecinos, esta información será almacenada en forma de tablas y será la que se transmita a los nodos vecinos, esta información servirá para establecer el enrutamiento entre los nodos, lo cual proporcionara una red independiente de los niveles inferiores de cara al usuario, de tal manera que un usuario pueda establecer conexiones de red con cualquier nodo de la red ad-hoc independientemente de si son vecinos o no. Para controlar el enrutamiento y la transmisión de la información necesitamos realizar el control de los vecinos , y una interacción con el modulo de autoconfiguración, y la sincronización de este, para conseguir que el traspaso de información y de enrutamiento se realice siempre entre nodos de la misma red ad-hoc, nuestro proyecto solo describe la necesidad de la integración de todos los módulos, dado que el funcionamiento de cada modulo depende del funcionamiento de los demás, y se limita a implementar el modulo de enrutamiento, quedara para el futuro la integración de todos los módulos, este modulo es independiente del modo de transmitir los datos a los nodos vecinos, para compartir información, serán los módulos inferiores quien se encargaran de esto.

El modulo de transmisión se encarga de transmitir la información, a nivel de red, este modulo no conoce el tipo de información que se transmite, serán los módulos superiores quien se encargaran de conocer la información que se intercambia, este modulo tiene dos partes fundamentales la de transmisión y la de recepción.

Los paquetes que se transmiten tienen un formato propio establecido por nosotros, son paquetes UDP de tamaño fijo, el puerto elegido para la comunicación ha sido el 2323 ya que es un puerto no utilizado por ninguna aplicación conocida,cada mensaje contendrá 10 entradas de la tabla de rutas, cada una con el destino, el gateway para ese destino, y la distancia en saltos al nodo destino, para la distancia hemos reservado 1 byte por lo que podemos

admitir distancias de hasta 254 saltos y 255 se consideraría infinito, si necesitamos enviar mensajes con un menor tamaño de 10 entradas, se rellenaran las demás como gateway “0.0.0.0” destino “0.0.0.0” y no serán tomadas en cuenta, hemos elegido el tamaño de 10 entradas por paquete por que en un entorno wireless ad-hoc normal, el numero de 10 nodos vecinos por nodo, nos parece bastante razonable, de tal manera que se limitara la sobrecarga de las cabeceras, si se tuviera que enviar un numero mayor de entradas se encapsularían en varios paquetes, con esta estructura ocupamos 90 bytes por paquete, a continuación mostramos la estructura de los paquetes:




### 4.2.2 *Descripción de los módulos involucrados de enrutamiento*

**Transmisión de la información**

Este modulo es el encargado de enviar/recibir la información necesaria para el enrutamiento entre los nodos vecinos, aunque estos módulos son independientes del uso que se haga, ya son genéricos, pueden ser usados para transmitir/recibir cualquier tipo de mensajes UDP, a cualquier destino a cualquier puerto, este modulo tiene 2 partes fundamentales, la que se encarga de enviar paquetes bajo petición del usuario, y la que esta recibiendo paquetes constantemente, es fundamental que estos dos submodulos funcionen concurrentemente para el correcto funcionamiento de los algoritmos de cálculo de rutas, la concurrencia la conseguiremos mediante la utilización de posix threads.

**Submodulo de envió de paquetes**

Este submodulo esta formado por el archivo envia.c , el cual tiene una única función con la siguiente cabecera

```
int envia(char* contenido,int puertoDST,char* interfaz,char* dir_dst, char* dir_fuen)
```

Los argumentos son los siguientes

Contenido: array de caracteres que se corresponde con la carga del paquete

puertoDST: puerto destino del paquete

interfaz: interfaz del sistema desde la que deseamos enviar el paquete

dir_dst: dirección a la que va destinado el mensaje

dir_fuen: dirección desde la que enviamos el mensaje

El envio lo realizamos utilizando la librería libnet a continuación mostramos como se realiza la formación del paquete y el envio de este, los siguientes pasos han sido los que hemos seguido en esta función:

1. Creación del manejador de libnet con libnet_init
2. Creación del paquete UDP con libnet_build_udp, aquí añadimos el puerto destino y la carga del mensaje.
3. Construcción del paquete IP con libnet_build_ipv4
4. Envio del paquete con libnet_write
5. Liberar la memoria reservada para la creación de paquetes con libnet_destroy

Tras la realización de estos pasos el paquete se envía a la red por la interfaz indicada.

**Submodulo de recepción de paquetes**

Este modulo es el encargado de recibir los paquetes involucrados en el enrutamiento, este modulo debe de estar escuchando paquetes continuamente, para ello hemos utilizado la librería libpcap, además necesitaremos usar threads para que se ejecute concurrentemente el resto de tareas del enrutamiento, también necesitamos que se comunique con los demás módulos para que estos conozcan los paquetes que llegan y así poder modificar las tablas de la manera adecuada, esta comunicación la realizamos volcando el contenido de cada paquete a un archivo para que luego pueda ser procesado por el modulo de gestión de la información, con esto conseguimos una

comunicación asíncrona entre ambos módulos de tal manera que cada uno puede generar o procesar información independientemente.

A continuación analizamos el archivo moduloCaptura.c que será el encargado de la recepción de paquetes, este modulo esta compuesto por 3 funciones, captura() que será la encargada de realizar el proceso de inicialización de libpcap, my_callback, que será la función a la que se llamara cada vez que se reciba un paquete, también esta la función dump que será la encargada de extraer la carga del paquete y escribirla en un archivo, a continuación detallamos el funcionamiento de cada una de estas funciones:

**Capura():** como hemos comentado anteriormente esta función se encarga de inicializar libpcap para la captura, finalmente se llama al bucle de captura para que este continuamente capturando paquetes los pasos seguidos son los siguientes:

1. buscar la interfaz de red del sistema usando pcap_lookupdev
2. abrir esa interfaz en modo captura pcap_open_live
3. establecer el filtro de captura de paquetes, en nuestro caso capturaremos paquetes UDP por el puerto 2323
4. Compilar el filtro con pcap_compile
5. Aplicar el filtro con pcap_setfilter
6. Llamar al bucle de captura con pcap_loop

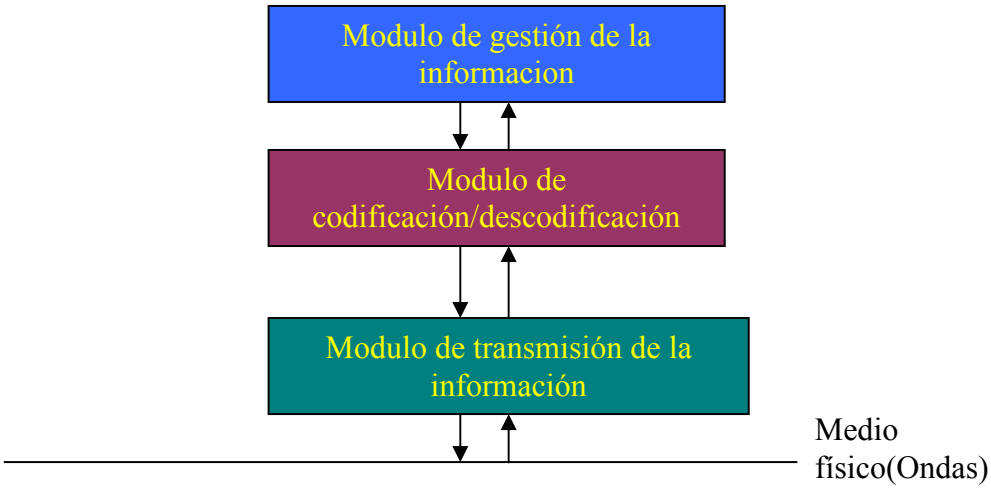
**my_callback():** Esta función es la que se ejecuta cada vez que llega un paquete, es la encargada de crear el archivo en el que se guardara el paquete, el archivo tiene un nombre con una componente numérica incremental de tal manera que se puedan procesar en el mismo orden en el que llegan, primero se escribe la ip del emisor del mensaje, y posteriormente en cada línea escribirá en forma de string un numero decimal que se corresponderá con un byte.

**dump():** Esta función es la que se encarga de extraer la carga del paquete y escribirla en el fichero

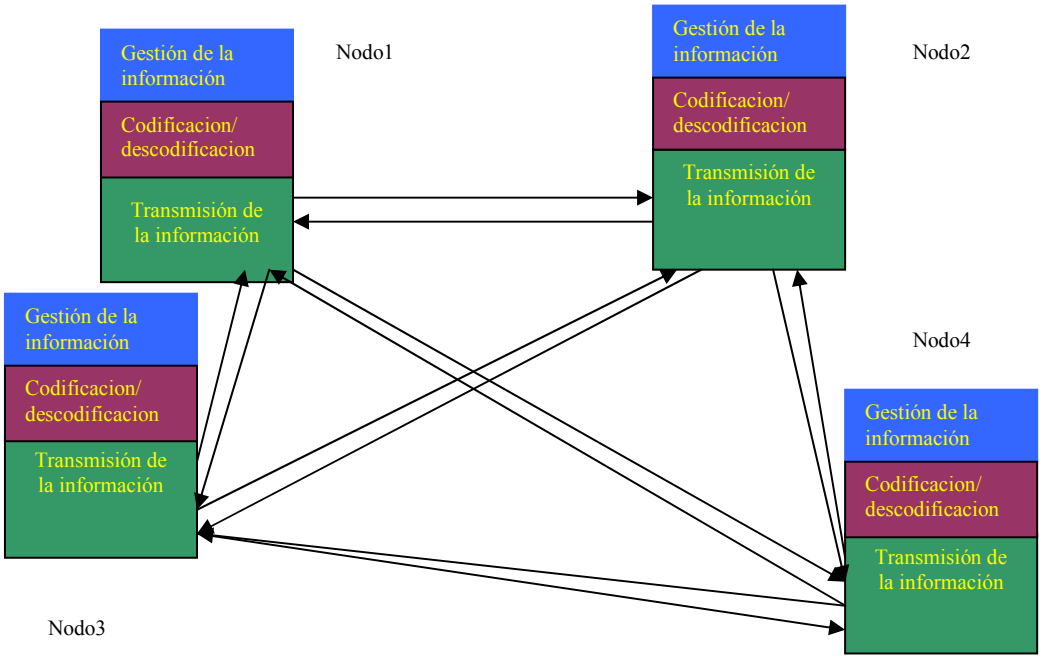
**Modulo de codificación/descodificación**

Aunque en la visión mas abstracta de nuestro sistema hablábamos simplemente de modulo de gestión de la información y modulo de transmisión de la información, necesitamos que ambos módulos se comuniquen, ya que el modulo de gestión de la información, a la hora llamar a las funciones de transmisión envía la información tal y como la maneja el, sin saber como se enviara en los módulos de transmisión/recepción, por ello es necesario un nivel intermedio que se encargue de interpretar los datos del modulo de gestión y encapsularlos de tal manera que puedan ser tomados por las funciones de transmisión/recepción, en nuestro sistema este modulo esta compuesto por los archivos codifica.c y decodifica.c y recupera.c

Teniendo en cuenta esto el sistema de enrutamiento de cada nodo quedaría de la siguiente manera




El entorno de trabajo compuesto por varios nodos quedaría de la siguiente forma:




A continuación pasamos a describir el funcionamiento de cada uno de los archivos c, implicados en este submodulo.

**Recupera.c :**

Como hemos comentado anteriormente cada vez que se recibe un paquete UDP al puerto 2323 se procesa y se escribe el contenido en un archivo

para su posterior proceso, pero antes de procesarlo debemos extraerlo del archivo. Este es el cometido de este archivo, tiene 3 funciones para extraer el contenido del archivo, son las siguientes:

```
Int*  getPayload(char* nombre,int* mensaje)
```

Esta función se encarga de extraer la carga del mensaje, tomara como entrada el nombre del fichero y devolverá el contenido de la carga en un array de enteros los parámetros son los siguientes

Nombre : es el nombre del archivo a extraer

Mensaje. Es el array de enteros donde se devolverá el contenido

```
char*  getDst(char* nombre,char* destino)
```

Esta función devuelve el destinatario del mensaje, donde nombre es el nombre del archivo y destino es el string donde devolverá la ip del destino.

```
char*  getSrc(char* nombre,char* fuente)
```

Esta función devuelve el destinatario del mensaje donde nombre es el nombre del archivo y fuente es el string donde devolverá la ip del emisor del mensaje.

**Codifica.c**

La finalidad de este archivo es transformar las entradas en la tabla de rutas del formato que utiliza el modulo de gestión de la información, al formato que utiliza el modulo de transmisión de la información, Este archivo se encarga de crear un array de enteros e ir almacenando entradas en la tabla de rutas bajo petición del modulo de gestión de la información finalmente se transforma ese array de enteros a un array de caracteres que coinciden con el carácter ASCII correspondiente a cada entrada en el array, ya que el modulo de envio solo es capaz de enviar arrays de caracteres, las funciones que tiene este archivo son las siguientes:

```
struct mensaje* creaMensaje()
```

Esta función crea una estructura mensaje que esta compuesta por un array de 90 enteros (la máxima capacidad de un mensaje) y un entero que indica la ocupación del mensaje (numero de entradas).

```
struct mensaje* incluyeEntrada(struct mensaje* men,char* destino,char* gateway,int distancia)
```

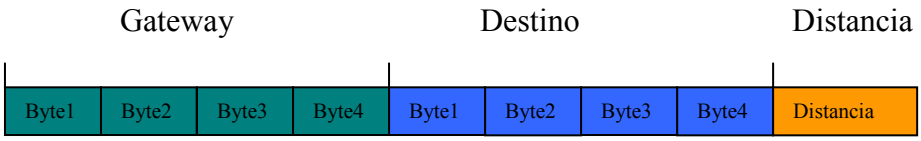
Esta función dada una estructura formada por una array de enteros, añade a esta una nueva entrada compuesta por un string que corresponde con el destino, otro string que se corresponde con el gateway y un entero que es la distancia entre ambos.

```
char* mensajecodificado(struct mensaje* men,char* codif)
```

Esta función se encarga de transformar la estructura compuesta de un array de enteros a un array de caracteres que es lo que tomara como entrada el modulo de envio, simplemente hace un casting de entero a unsigned char, para cada componente del array.

**Decodifica.c:**

El cometido de este archivo es recibir el contenido de un mensaje como un array de enteros, transformarlo en un array de estructuras que serán mas fáciles de manejar por el modulo de gestión de la información, el mensaje llega procedente de la función que se encarga de recuperar el mensaje del archivo y lo empaqueta en un array de enteros, donde en cada posición se encuentra el byte de una ip, primero se sitúa la ip del gateway para esa entrada, luego la ip del destino y por ultimo la distancia el esquema de una entrada seria el siguiente




La salida de este modulo es una array de estructuras, correspondiéndose cada estructura con una entrada en la tabla de rutas, esta estructura esta compuesta por 2 strings que se corresponden con el destino y el gateway de la entrada y un entero que se corresponde con la distancia:

```
struct entrada{

      char destino[20];

      char gateway[20];

      int distancia;

};
```

La función que realiza la extracción del mensaje en un array de estructuras es:

```
struct entrada* decodifica(struct entrada* tabla,int* mensaje)
```

donde:

tabla es el array de entradas al que se transforma el mensaje

mensaje es el array de enteros que se corresponde con el mensaje tal y como se extrae del archivo.

Este archivo tiene una función auxiliar decoIP que dado un array de enteros y una posición en el array, transforma las 4 siguientes posiciones en un string que se corresponde con una dirección ip con la forma “x.x.x.x”, esta función nos servirá para transformar las direcciones ip del array de enteros a un string, la función es la siguiente:

```
char* decoIP(char* ip, int* mensaje,int i)
```

Donde :

ip es el string donde devolverá la ip extraída;

i es la posición del array donde empieza la ip

mensaje es el array de enteros que se corresponde con el mensaje

**Modulo de gestión de la información**

Como hemos comentado anteriormente este modulo es quien realizara el algoritmo especificado por RIP, para ello se ayudara de los otros módulos para enviar y recibir paquetes, este modulo tiene 2 partes fundamentales tablas.c que define las estructuras de datos para las tablas de enrutamiento y las operaciones sobre estas tablas, la otra componente del modulo de gestión de la información es enrutamiento.c el cual se encarga de implementar el algoritmo RIP, a continuación mostramos el funcionamiento de cada uno de estos archivos

**Tablas.c**

Este archivo define las estructuras de datos para la tabla de vecinos y para la tabla de enrutamiento, estas estructuras son las siguientes:

Para la tabla de rutas: la tabla de rutas esta compuesta por un array de entradas a dicha tabla un índice a la siguiente posición libre en la tabla, un entero que indica el numero de entradas que tiene la tabla, y otro entero que indica la ocupación de la tabla, esta estructura se define de la siguiente manera:

```
struct truta {

      struct eTablaRuta* entradas;

      int libre;

      int size;

      int ocupados;

};
```

Cada una de las entradas de la tabla de rutas esta compuesta por los siguientes campos, un string que se corresponde con el destino de dicha entrada, un string que se corresponde con el gateway, un entero que es la distancia al destino, un entero que sirve de flag para saber si esa entrada esta realmente ocupada, u un temporizador para saber cuando se actualizo esa entrada(aunque este campo actualmente no se utiliza), también se almacena el nodo desde el que se aprendió esta ruta, para que en el proceso de actualización no se envié una entrada al nodo del que la hemos aprendido, de esta manera evitaremos inconsistencias, la estructura es la siguiente:

```
struct eTablaRuta {

      char destino[20];

      char gateway[20];

      char aprendido[20];

      int distancia;

      time_t temporizador;

      int ocupada;

};
```

Para la tabla de vecinos las estructuras que hemos utilizado han sido las siguientes:

La tabla de vecinos tiene un array de entradas a dicha tabla un índice a la siguiente posición libre en la tabla, un entero que indica el numero de entradas que tiene la tabla, y otro entero que indica la ocupación de la tabla, esta estructura se define de la siguiente manera:

```
struct tvecinos {

      struct eTablaVecinos* entradas;

      int libre;

      int size;

      int ocupados;

};
```

Cada entrada en la tabla de vecinos tiene los siguientes campos, un string que se corresponde con la ip del vecino en cuestión, un entero que sirve de flag para saber si la entrada esta ocupada, y un temporizador para saber cuando fue la ultima vez que recibimos información de este nodo de tal manera que cuando llevemos mucho tiempo sin recibir información de este nodo daremos por hecho que no esta a nuestro alcance y lo eliminaremos de la tabla

de vecinos ,otro temporizador para saber cuando fue la ultima vez que enviamos información a este nodo, la estructura quedaría de la siguiente forma:

```
struct eTablaVecinos {

char  vecino;

time_t recepcion;

time_t envio;

int ocupada;

};
```

Además de las estructuras de datos también se definen operaciones para trabajar con ellas,a continuación mencionamos las funciones de las cuales disponemos

**struct truta* creaTablaRutas(struct truta* tabl):**Esta función se encarga de crear la estructura de la tabla de rutas

**struct tvecinos* creaTablaVecinos(struct tvecinos* tabl):** Esta función se encarga de crear la estructura de la tabla de rutas

**struct truta* incluyeEntradaTRuta(struct truta* tabla, char* destino,char* gateway, char* aprendido,int distancia):** esta función se encarga de añadir una entrada en la tabla de rutas, se pasan como parámetros, el destino de la entrada el gateway la distancia y el nodo desde el que se aprendió la ruta, devuelve la tabla de rutas actualizada, utiliza la función buscaLibre para buscar la siguiente posición libre en la tabla.

**struct truta* incluyeEntradaTVecinos(struct tvecinos* tabla, char* vecino,time_t envio,time_t recibido):** Esta función se encarga de añadir una entrada en la tabla de vecinos, los parámetros que se pasan son la tabla donde queremos añadir la entrada, la ip del vecino y las dos marcas de tiempo, esta función utiliza la función buscaLibreTVecinos para buscar la siguiente posición libre en la tabla.

**int getDistancia(struct truta* tabla,char* destino):**Esta función devuelve la distancia que existe en la tabla de rutas hasta el nodo destino

**struct tvecinos* buscaLibreTVecinos(struct tvecinos* tabla):**Esta función busca la siguiente entrada libre en la tabla de vecinos.

**struct truta* buscaLibre(struct truta* tabla):** Esta función busca la siguiente entrada libre en la tabla de rutas.

**struct truta* eliminaEntradaTRuta(struct truta* tabla,char* destino):** Esta función busca un determinado destino en la tabla de rutas y lo elimina de ésta.

**struct tvecinos* eliminaEntradaTVecinos(struct tvecinos* tabla,char* vecino):** busca en la tabla de vecinos por la ip del campo vecino y elimina esa entrada de la tabla.

**struct truta* modificaEntradaTVecinos(struct tvecinos* tabla, char* vecino,time_t envio,time_t recibido): E**sta función busca una entrada en la tabla de vecinos que se corresponda con el campo vecino, y modifica sus atributos con los que se pasan como parámetros.

**struct truta* modificaEntradaTRuta(struct truta* tabla, char* destino,char* gateway, char* aprendido,int distancia):**Esta función busca una entrada en la tabla de rutas que se corresponda con el campo destino, y modifica sus atributos con los que se pasan como parámetros.

**int existeEntradaTVecinos(struct tvecinos* tabla,char* vecino):** comprueba en la entrada con el campo vecino igual al que se le pasa por parámetro.

**void imprimeTVecinos(struct tvecinos* tabla):**Esta función imprime por pantalla la tabla de vecinos.

**void imprimeTRuta(struct truta* tabla):**Esta función imprime por pantalla la tabla de rutas.

**enrutamiento.c:** Este archivo es el mas importante de nuestro sistema ya que es el que lleva toda la lógica del enrutamiento, a continuación mostramos el funcionamiento de este modulo, primero tenemos que tener en cuenta que debemos realizar 2 tareas fundamentales, enviar la tabla de rutas a los vecinos, procesar la información recibida, estas acciones se deben realizar de manera simultanea por ello necesitamos la utilización de threads, en nuestro caso

el thread que se procesara en background será el que se encarga de capturar paquetes y el hilo principal será el que realiza todo el control del enrutamiento, la distribución de los threads se hace de la siguiente manera:

```
void controlEjecucion() {

      pthread_t thread1;

      char *message1 = "Thread 1";

      int  pid;

      //creamos el thread que realiza la captura de los paquetes

      pid = pthread_create( &thread1, NULL, ProcesoDeCaptura,
      (void*) message1);
```

```
        controlEnrutamiento();

        pthread_join( thread1, NULL);

    }
```

Se crea el thread que ejecuta la función procesoDeCaptura que a su vez llama a la función correspondiente del modulo de captura, a continuación mostramos el proceso que se realiza en la función controlEnrutamiento.

En esta función es un bucle infinito en el que cada vuelta procesa un paquete de los que se reciben y controla que cada 30 segundos se envíen las tablas a los nodos vecinos , así como controlar la eliminación de los nodos vecinos de los que hace mas de 2 minutos que no se reciben actualizaciones

```
    while(1) {

                tiempo_actual=time(NULL);

                if((((int)tiempo_actual)-((int)ultimo_envio))>30)

                      {

                      enviaTablas(tablaRuta,tablaVecinos,mi_ip,ifaz);

                      time_t ultimo_envio=time(NULL);

                      }


    analizaPaquete(&contadorPaquetes,tablaRuta,tablaVecinos,mi_ip,if
    az);

                tablaVecinos=eliminaVecinos(tablaVecinos);

        }
```

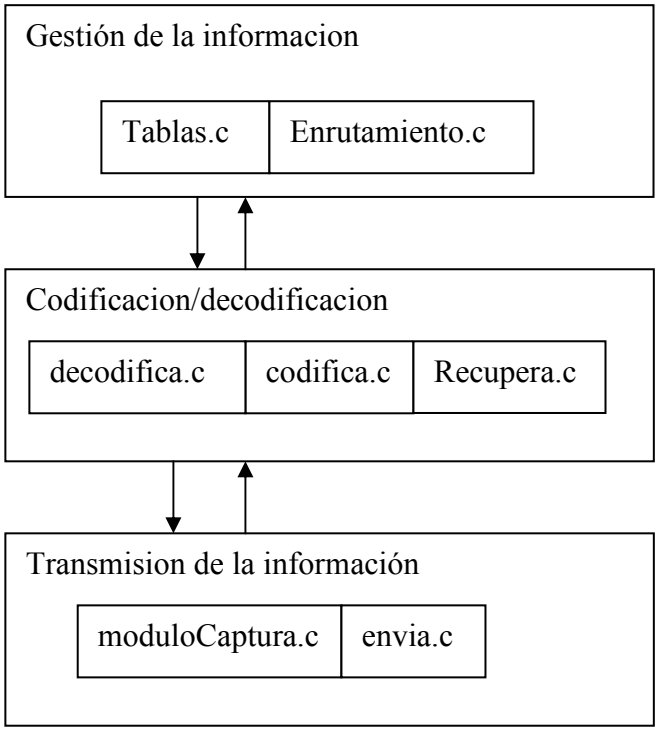
En la función analizaPaquete se hace un bucle de 1 a 10(que es el numero máximo de entradas por paquete), y para cada entrada se comprueba que no es una entrada vacía, de no serlo se comprueba si existía en la tabla de rutas, de no existir la anotamos, y si ya existiera comprobamos si la distancia es menor que la que había anteriormente anotamos la nueva entrada de lo contrario no haremos nada, si se produjera algún cambio en las tablas, se provocaría el envió de estas a los vecinos para evitar inconsistencias, a continuación adjuntamos el código de la función

```
    while(i<10) //para cada entrada del mensaje comprobamos que no
    esta vacia

    {

    if((strcmp(entradas[i].destino,"0.0.0.0")!=0)&&

      (strcmp(entradas[i].gateway,"0.0.0.0")!=0))
```

```
	{

	if(existeEntradaTRuta(entradas[i].destino)==0)

		{

//si la entrada no estaba en nuestra tabla de rutas creamos la
entrada nueva


tablaRuta=incluyeEntradaTRuta(tablaRuta,entradas[i].destino,fuen
te_mensaje, fuente_mensaje,entradas[i].distancia+1);

		}

	}

	else

		{



		int
distancia=getDistancia(tablaRuta,entradas[i].destino);

		if(entradas[i].distancia+1<distancia)

{//si la entrada nueva esta mas cerca que la antigua enviamos
las tablas a nuestros vecinos



tablaRuta=modificaEntradaTRuta(tablaRuta,entradas[i].destino,fue
nte_mensaje, fuente_mensaje,entradas[i].distancia+1);

		enviaTablas(tablaRuta,tablaVecinos,mi_ip,ifaz);
			}

	}

i++;

}
```

Las funciones de envió de las tablas toman la tabla de vecinos y para cada uno de ellos le envían todas las entradas en la tabla de rutas menos las que aprendió de ese mismo nodo, como el tamaño de los paquetes es de 10 entradas, en caso de tener un tamaño de la tabla de rutas mayor procederemos a fragmentarlo en paquetes de 10 entradas, finalmente llamamos al modulo de envió para mandar el paquete.

Una vez vistos detenidamente todos los modulos de nuestro sistema podemos ver que la estructura queda de la siguiente manera:




**Compilación**

Para compilar el código previamente se deben tener instaladas las librerías de posix threads, libpcap y libnet, a parte de tener el compilador gcc instalado, tras instalar estas librerías bastara con descomprimir nuestra aplicación y ejecutar el comando make para compilarlo.

# 5 Herramientas Utilizadas

- Sistema Operativo, Linux
- Lenguaje de Programación C
- Librería:
    - Libpcap
    - Libnet
    - Posix Threads

# 6 Glosario

- Ad Hoc
- Wifi
- Redes inalámbricas multi-salto
- Autoconfiguración
- Sincronización
- Enrutamiento
- DHCP
- RIPv2
- Libnet
- Libpcap

Bibliografía

- http://libnet.sourceforge.net/

- http://www.tcpdump.org/

- Aprendiendo a programar con Libpcap,Alejandro López Monge 20/2/2005

- http://www.humanfactor.com/pthreads/

- Evaluación de Propuestas de Interconexión a Internet para Redes Móviles Ad Hoc Híbridas

-